Towards Secure Outsourced Data Services in the Public Cloud

Wenhai Sun

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Application

June 21, 2018
Falls Church, Virginia

Keywords: Cloud Computing, Privacy-preserving Keyword Search, Verifiable Computation, Secure Genomic Computation, Secure Data Deduplication, Trusted Hardware

Towards Secure Outsourced Data Services in the Public Cloud

Wenhai Sun

(ABSTRACT)

Past few years have witnessed a dramatic shift for IT infrastructures from a self-sustained model to a centralized and multi-tenant elastic computing paradigm – Cloud Computing, which significantly reshapes the landscape of existing data utilization services. In truth, public cloud service providers (CSPs), e.g. Google, Amazon, offer us unprecedented benefits, such as ubiquitous and flexible access, considerable capital expenditure savings and on-demand resource allocation. Cloud has become the virtual "brain" as well to support and propel many important applications and system designs, for example, artificial intelligence, Internet of Things, and so forth; on the flip side, security and privacy are among the primary concerns with the adoption of cloud-based data services in that the user loses control of her/his outsourced data. Encrypting the sensitive user information certainly ensures the confidentiality. However, encryption places an extra layer of ambiguity and its direct use may be at odds with the practical requirements and defeat the purpose of cloud computing technology. We believe that security in nature should not be in contravention of the cloud outsourcing model. Rather, it is expected to complement the current achievements to further fuel the wide adoption of the public cloud service. This, in turn, requires us not to decouple them from the very beginning of the system design.

Drawing the successes and failures from both academia and industry, we attempt to answer the challenges of realizing efficient and useful secure data services in the public cloud. In particular, we pay attention to security and privacy in two essential functions of the cloud "brain", i.e. data storage and processing. Our first work centers on the secure chunk-based deduplication of encrypted data for cloud backup and achieves the performance comparable to the plaintext cloud storage deduplication while effectively mitigating the information leakage from the low-entropy chunks. On the other hand, we comprehensively study the promising yet challenging issue of search over encrypted data in the cloud environment, which allows a user to delegate her/his search task to a CSP server that hosts a collection of encrypted files while still guaranteeing some measure of query privacy. In order to accomplish this grand vision, we explore both software-based secure computation research that often relies on cryptography and concentrates on algorithmic design and theoretical proof, and trusted execution solutions that depend on hardware-based isolation and trusted computing. Hopefully, through the lens of our efforts, insights could be furnished into future research in the related areas.

Towards Secure Outsourced Data Services in the Public Cloud

Wenhai Sun

(GENERAL AUDIENCE ABSTRACT)

Past few years have witnessed a dramatic shift for IT infrastructures from a self-sustained model to a centralized and multi-tenant elastic computing paradigm – Cloud Computing, which significantly reshapes the landscape of existing data utilization services. In truth, public cloud service providers (CSPs), e.g. Google, Amazon, offer us unprecedented benefits, such as ubiquitous and flexible access, considerable capital expenditure savings and on-demand resource allocation. Cloud has become the virtual "brain" as well to support and propel many important applications and system designs, for example, artificial intelligence, Internet of Things, and so forth; on the flip side, security and privacy are among the primary concerns with the adoption of cloud-based data services in that the user loses control of her/his outsourced data. Encryption definitely provides strong protection to user sensitive data, but it also disables the direct use of cloud data services and may defeat the purpose of cloud computing technology. We believe that security in nature should not be in contravention of the cloud outsourcing model. Rather, it is expected to complement the current achievements to further fuel the wide adoption of the public cloud service. This, in turn, requires us not to decouple them from the very beginning of the system design. Drawing the successes and failures from both academia and industry, we attempt to answer the challenges of realizing efficient and useful secure data services in the public cloud. In particular, we pay attention to security and privacy in two essential functions of the cloud "brain", i.e. data storage and processing. The first part of this research aims to provide a privacy-preserving data deduplication scheme with the performance comparable to the existing cloud backup storage deduplication. In the second part, we attempt to secure the fundamental information retrieval functions and offer effective solutions in various contexts of cloud data services.

To the three important women in my life,
my mother Lingxia, my wife Ying, and my daughter Emily.

Acknowledgments

Two Ph.D.s are earned during a period of seven years, which I would never imagine when I was an undergraduate student at Xidian University back in China. I clearly and vividly remember how this unbelievable journey started. I will not drive you into my biography, which I believe is too early for my age, but I would like to express my greatest gratitude to all the people who work with, help, encourage or inspire me along this journey.

I have received countless help from many people during this dissertation research, but it would not be possible without the support and mentoring of my advisor, Dr. Wenjing Lou. She showed me how interesting and rewarding the research could be during my short-term visit to Virginia Tech. That is why I decided to pursue my second Ph.D. under her guidance. Not only does she reshape my research, she also reforms my life and the way I am looking at this world. Words cannot express how grateful I am to her. I am also thankful to Dr. Y. Thomas Hou. Albeit we do not spend much time working together due to the long distance, it does not change the fact that you are another role model of mine. Your vision, determination, execution, insight, and leadership all have been inspiring me throughout my Ph.D. study. In addition, I would like to thank other committee members, Dr. Ing-Ray Chen, Dr. Gang Wang, and Dr. Yanchao Zhang, for your full support, invaluable comments and suggestions. I am also privileged to collaborate with many talented researchers, and receive tons of help from people around me, including (not exhaustively and no particular order): Dr. Ning Zhang, Dr. Ming Li, Dr. Shucheng Yu, Dr. Xuefeng Liu, Dr. Ning Cao, Ruide Zhang, Dr. Bing Wang, Dr. Qiben Yan, Dr. Yao Zheng, Dr. Changlai Du, Yang Xiao, and Yaxing Chen. Thank you all.

In the end, I am deeply indebted to my family. Mom, I owe you a big thanks for devoting yourself to taking care of me and protecting me when I was little and sensitive. You have been supporting and having faith in me all the time. I am hopeful what I have accomplished makes you proud. Ying, you are the most wonderful girl I have ever met. You are the reason that I stay optimistic and confident, and keep chasing my dreams even when going through ups and downs. I am grateful for your sacrifice in every respect. I am fortune that I will spend the rest of my life with Keaibao. Emily, my daughter, you are smart, joyful and always kind to people around. You are the best present I have ever had. I am always surprised by the amazing power within your small body that changes my life silently, positively and significantly.

Funding Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation grants 1217889, 1405747, 1446478 and 1443889.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are in a big-data era, which brings us a tremendous amount of information recording incredible details of our world and life. We wholeheartedly embrace it because of the potentially huge value of this information treasure that is expected to advance the human society and improve our life quality. On the other hand, storing and processing the data requires IT practitioners to constantly purchase hardware and software, and provide frequent corresponding personnel training. This has become a significant burden that possibly offsets the promising benefits. Fortunately, past few years have witnessed a dramatic shift for IT infrastructures from a self-sustained model to a centralized and multi-tenant elastic computing paradigm – Cloud Computing. People are moving their local digital assets and applications to the large data center operated by a third-party CSP, e.g. Google, Amazon, in order to be relieved from the cumbersome IT management and enjoy considerable capital expenditure savings. Besides, cloud also offers other benefits, such as ubiquitous and flexible access, on-demand resource allocation, which further expedites and contributes to the prevalence of this data outsourcing model.

1.1 Securing Data Services in Cloud Computing

While cloud has gained momentum and become the "brain" in the present networked information ecosystem to support and propel many important applications and system designs, including business and productivity apps, artificial intelligence, Internet of Things, and so forth, security and privacy are among the primary concerns when using cloud-based data services in that the user hands over the control of her/his data to the public cloud that has not fully won the trust of the general public. Encrypting the sensitive user information certainly ensures the confidentiality. However, encryption places an extra layer of ambiguity and its direct use may be at odds with the practical requirements and defeat the purpose of cloud computing. We believe that security in nature should complement the current achievements to further fuel the growth of the public cloud services, which, in turn, requires us not to decouple them from the very beginning of the system design. The cloud brain, akin to a human brain, provides two essential functions – data storage and processing. In this dissertation, we aim to secure these fundamental data services in the challenging public cloud infrastructure as shown in Figure. 1.1.

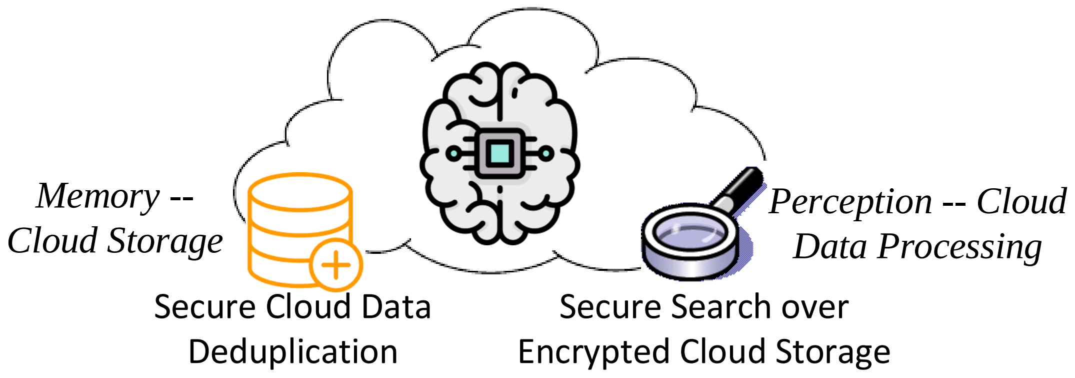


Figure 1.1: The studied secure cloud services.

1.1.1 Chunk-based Encrypted Cloud Data Deduplication

We first consider the cloud storage an equivalent of the memory function of the human brain. In order to save storage and/or network bandwidth, current cloud storage providers, such as Dropbox, Google Drive, exploit the data deduplication technique to effectively identify and eliminate redundant data. As a result, the system only keeps a single copy of the same files and makes a reference pointing to the stored copy for other duplicates. Data deduplication will remove more redundant information when applied to small chunk level than the file level, and thus often lead to higher storage savings. In Chapter 2, we first investigate the problem of designing a secure chunk-based deduplication scheme in the enterprise backup storage setting [112]. Most of the research in the literature focus on realizing file-based encrypted data deduplication or key/metadata management. Little attention is drawn to the practical chunk-level deduplication system. In particular, we identify that the information contained in a small-sized chunk is more susceptible to the brute-force attack compared with the file-based counterpart. We propose a *randomized oblivious key generation* mechanism based on the inner workings of the backup service. In contrast to the current work that compromising one client would eventually expose all the clients' storage, our scheme offers a counter-intuitive property of achieving *security against multi-client compromise* with *minimal deduplication performance loss*. In addition, we enforce a *per-backup* rate-limiting policy to slow down the online brute-force attack. We show that the proposed scheme is provably secure in the malicious model. We calibrate the system design as well by taking into account the requirements in reality to accomplish a comparable plaintext deduplication performance.

Our experiment on the real-world dataset shows its efficiency, effectiveness, and practicality.

1.1.2 Search over Encrypted Cloud Data

Perception is another main human brain function closely interacting with the memory. Similarly, we also pay a great amount of attention to the secure data processing in the public cloud. In particular, we comprehensively study the problem of search over encrypted data (SE), which allows the cloud to offer fundamental information retrieval (IR) service to its users in a privacy-preserving way. The IR research is relatively mature in plaintext domain, which is extensively used and embedded in many applications. As a critical building block of an information processing system, plaintext IR provides a wide range of functions and query types, which enables its users to effectively retrieve the intended information from a target dataset. However, its merits of flexibility, efficiency, robustness, and function-diversity will be significantly diminished in the ciphertext world. Once we exploit cryptography to encrypt everything and then store them in the cloud, how can we apply existing IR functions later to the random-looking ciphertext that hide all the semantic details, and expect to receive the correct result? Even if we use fully homomorphic encryption [46] that generally is deemed the panacea for computation over encrypted data, you are likely to have a non-negligible performance hit for the time being. As a matter of fact, there is no silver bullet for SE. With the ultimate research goal of building efficient and practical SE framework, we propose three SE schemes centering on different contexts and design aspects by taking advantage of software and hardware-based approaches. We make contributions to both SE theory and real-world applications and implementations.

Verifiable Conjunctive Keyword Search

The search result in most SE schemes is returned by a *honest-but-curious* or *semi-trusted* server that executes the protocol faithfully but curiously infers the user information en route. Under this threat model, we usually consider the query result to be correct. However, there may exist a malfunctioning server or even a *malicious* adversary that is able to arbitrarily deviate from the designated protocol in practice. Therefore, users need a result verification mechanism to detect the potential misbehavior and rebuild the confidence for the outsourced search operation. Further, cloud typically hosts a large amount of outsourced data of users in its storage. The verification complexity should be low enough for practical use. In other words, it is desired to only depend on the corresponding search operation, regardless of the file collection size. Otherwise, the users might as well search the data by themselves. In Chapter 3, we introduce our first SE work to investigate the efficient search result verification [109]. We target the common query type – conjunctive keyword search, i.e. searching for a combination of multiple keywords. In addition, the proposed SE scheme enables users to freely update the secure index and the corresponding file collection. The presented verification mechanism can be either delegated to a *public* trusted authority (TA) or be executed *privately* by data users. We formally prove the *universally composable* (UC) security of our scheme. Experimental result demonstrates practicality even with a large dataset.

Secure Genome-wide Range Query

We in Chapter 4 deal with a real-world SE application in the context of the promising human genomic/medical data research [111]. As the cost of full genome sequencing technology continues to drop, we will soon witness the proliferation of human genomic data in the public cloud. In order to protect the confidentiality of the sensitive genetic information of individuals, the stored data are preferred to be encrypted. However, as stated early, encryption severely hinders the use of this valuable information, such as Genome-wide Range Query (GRQ), in medical/genomic research. The examples of GRQ include that a pharmaceutical company issues a query on a particular range of the genomic data of a patient in order to find some DNA fingerprints/biomarkers for personalized medicine. While the problem of secure range query on outsourced encrypted data has been extensively studied, the current solutions are far from practical deployment in terms of efficiency and scalability due to the huge size of the sequencing result. We study the problem of secure GRQ over human raw aligned genomic data in a third-party cloud. Our solution contains a novel privacy-preserving range query design based on multi-keyword symmetric searchable encryption (MSSE). The proposed scheme incurs minimal ciphertext size expansion and computation overhead. We also present a hierarchical and GRQ-oriented secure index structure tailored for efficient and large-scale genomic data lookup in the cloud while preserving the query privacy. Our experiment on real human genomic data shows that a secure GRQ request with the range size of 100,000 over more than 300 million encrypted short reads takes less than 3 minutes, which is orders of magnitude faster than the state of the art.

Secure Keyword Search Using Trusted Hardware

We share the belief that heeding only one side will be benighted. Majority of the existing SE schemes, including our previous work, are software-based solutions built on top of diverse cryptographic primitives, which result in a rich set of secure search indexes and algorithm designs. However, each such SE solution can only implement a small subset of IR functions and often leaks considerable private search information. Recently, the hardware-based secure computation has emerged as an effective mechanism to securely execute programs in an untrusted software environment. Our third SE work in Chapter 5 exploits the hardware-based trusted execution environment (TEE) and explore a software and hardware combined approach to address the challenging SE problem [113]. For functionality, our design can support the same spectrum of plaintext IR functions. For security, we present oblivious keyword search techniques to mitigate the index search trace leakage. We build a prototype of the system using Intel SGX. We demonstrate that the proposed system provides broad support for a variety of search functions and achieves computation efficiency comparable to plaintext data search with elevated security protection. Hopefully, through the lens of this effort, insights could be provided into future research in this direction.

1.2 Research Contributions

In this dissertation, we study the protection of user data services in the public cloud environment and have made the following major research contributions.

For privacy preservation in the cloud storage deduplication:

- To the best of our knowledge, we are among the first to discuss the challenges of and solutions to securing chunk-based deduplication of encrypted backup storage as per the practical performance requirements. We propose a randomized oblivious key generation algorithm, which can effectively reduce the risk of the information leakage by being resilient to multiple compromised clients. We also enforce a per-backup rate limiting policy to slow down the online brute-force attack. Our presented scheme is provably secure in the malicious model. We show that the efficiency of the online key generation for frequent insensitive data can be significantly improved by using the content-aware deduplication technique. The experiment on the real-world dataset demonstrates a faster data restore speed while retaining an on-par deduplication effectiveness for backup storage compared to the plaintext chunk-based deduplication.

For securing cloud data search function:

- Our first SE work supports *conjunctive keyword search*, *dynamic data update* and *search result verification* simultaneously. We evaluate the performance of the scheme with a large real-world dataset and show that it is efficient enough for practical use. The verification cost merely depends on the corresponding search operation, irrespective of the size of the searched data collection. Furthermore, the verification mechanism is flexible in the sense that it can be either delegated to a *public* trusted authority or be executed *privately* by a data user. We also formally prove that our proposed scheme is *UC-secure* against a *malicious* adversary.

- In the second SE research work, we propose a novel secure range query scheme by designing a multi-keyword symmetric searchable encryption, which may be of independent interest. It is suitable for search over the space-consuming raw genomic data storage. The security of the proposed scheme is derived from the MSSE security against adaptive chosen-keyword attacks (CKA2) [34]. Our scheme shows the same privacy guarantees as the current practical solution but with a much better (logarithmic-time) query efficiency. By plaintext ordering obfuscation and a privacy-preserving re-sorting technique, we present a hierarchical secure index structure, which captures the real-world situation of genomic data processing in the cloud, featuring a scalable and efficient GRQ search over human raw aligned genomic data. We implement our proposed scheme on real human sequencing data. The experiment demonstrates the efficiency and its promising future deployment in a large-scale GRQ scenario.

- We propose secu**r**e k**e**yword se**ar**ch usin**g** tr**u**sted h**ard**ware – REARGUARD, an innovative approach towards a dynamic secure keyword search scheme that employs the latest advancement in hardware-based trusted execution – Intel SGX. The proposed system supports a rich set of IR functions and query types while ensuring the confidentiality and integrity of the query process. Our design is secure against strong attacks including those from the compromised privileged software and low-level firmware. Our design mitigates the index search trace leakage from the memory side channel by using oblivious keyword search functions. This is one of the major concerns about SE in the cloud where users and adversaries share resources. The system defines and realizes two leakage profiles to balance security and performance, both exhibiting substantially reduced index search footprints. We carefully design and implement the popular IR functions into a fully-functional SGX-compatible prototype. Our experiment with the real-world dataset reports a performance close to that of plaintext data search with elevated security protection.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 presents our work on secure chunk-based deduplication of encrypted data for cloud backup. Chapter 3 deals with the efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In Chapter 4, we describe the design and validation of a scalable and efficient range query scheme over encrypted human raw alignment data. Chapter 5 introduces our novel secure keyword search scheme based on trusted hardware. Finally, Chapter 6 summarizes the research achievements and the potential future research directions.

Chapter 2

Secure Chunk-based Deduplication of Encrypted Data for Cloud Backup

2.1 Introduction

Data deduplication (or dedupe for short) is increasingly adopted by cloud storage providers, such as Amazon S3, as an effective technique to reduce the storage cost. When the system detects data redundancy, dedupe process will retain only one copy of the same data and make a reference pointing to the stored copy for other duplicates. Data confidentiality is realized by exploiting deterministic encryption, e.g., convergent encryption (CE) [39], in which the key is generated from the data itself and the same plaintext will always yield the same key and ciphertext. As a result, we can apply dedupe to the ciphertext without leaking underlying stored information. However, the existing secure deduplication designs [10, 39, 40, 72, 90, 103], to some extent, are at odds with the real-world dedupe requirements in terms of security and performance.

2.1.1 Knowing the Gap

By using different chunking methods, deduplication can be carried out either in the *coarse-grained* file level, or *fine-grained* chunk level. In a nutshell, file-based deduplication (FBD) is suitable for *small* or *stationary* file types, e.g., dll, lib, pdb, etc. Even small changes made in the file will lead to a completely different copy, thereby resulting in a low dedupe performance. In contrast, chunk-based deduplication (CBD) is capable of dividing a given data stream into smaller chunks of fixed or variable lengths (typically from 4KB to 16KB). As such, a considerable storage saving can be reaped from chunk-level redundancy elimination. In fact, there is a trend towards large files being the principal consumer of the storage [78]. More and more real-world storage systems are using CBD as their core deduplication technique [87].

Most of the existing works focus on secure file-level dedupe. Chunk-level designs [29, 68, 90] with different research concentration paid little attention to the challenges and practical requirements of CBD.

1) Low-entropy chunks. Deterministic CE is inherently vulnerable to brute-force attack for predictable files. For example, given the ciphertext, the adversary is able to enumerate all the file candidates, encrypt and compare them with the target ciphertext in an *offline* manner. Prior works provide solutions by deriving the encryption key from an *online* third party, i.e., either an independent key server [10] or other peer clients [72], instead of offline key generation from data itself. However, an adversary compromising one authorized client and observing the dedupe process can still launch an *online* brute-force attack. In general, the data leakage covers the storage of all clients in the system. A rate-limiting strategy may be enforced to slow down the attack speed. But such approach is merely effective if the deduplicated data has enough unpredictability. CBD will amplify the attack efficacy due to the potentially much lower entropy contained in a small chunk. Albeit it is an open problem to entirely prevent the brute-force attack, we still need to answer the question: *To what extent, can we reduce the risk of the information leakage with minimal impact on the underlying deduplication routine?*

2) Increased system operation overhead. Besides the inevitable cost of performing file chunking, directly applying existing schemes to chunk-level deduplication usually incurs higher latency and computation overhead. This is because the client needs to run the key generation protocol with other online parties (a key server [10] or peer clients [72]) to produce a CE key for each chunk of a file instead of one protocol execution for the whole file. Thus, a natural question is: *Can we speed up the key generation while still ensuring an effective deduplication function?*

3) Practical dedupe performance. In addition to the *deduplication ratio* (or *space reduction percentage*, see Section 2.2) that is widely used in measuring the effectiveness of deduplication [41], there are also other metrics in practice to determine the dedupe system performance, such as *chunk fragmentation level*, or *data restore speed* (see Section 2.2). Chunk fragmentation is caused by CBD in which data logically belonging to a recent backup scattered across multiple older backups [61]. A higher chunk fragmentation level typically adversely affects the system read performance and further increase the data restore cost. On the contrary, fragmentation is not widespread in file-based deduplication owing to the sequentially stored files on disk. It is expected that *any secure chunk-level dedupe design should provide a read performance on par with plaintext CBD practice.*

Aiming to answer the above challenges, we design a chunk-based deduplication scheme for encrypted enterprise backup storage in the cloud. The core of the technique is the proposal of a *randomized oblivious key generation* (ROKG) protocol, which is simple by its design but powerful by its efficacy. Specifically, we are inspired by the observation that using randomized encryption will completely protect the low-entropy chunks albeit it, in turn, will outright incapacitate the deduplication. Similarly, we attempt to introduce the randomness into the chunk key generation. This gives us the desired asymmetry between security and performance, i.e. *resilient to multiple compromised clients*, compared to existing work, but only with *minimal dedupe performance loss.* We confine the proposed security and privacy preservation design to the enterprise internal network via setting up a key server in order to stay transparent to and compatible with the existing public cloud backup service. We further accelerate the key generation for frequent insensitive data by leveraging the *content-aware* deduplication. A *per-backup* rate-limiting strategy is also presented to further slow down the online brute-force attack without interfering the dedupe procedure. In addition, our design achieves *faster data restore speed* and *comparable space savings* for backup storage with plaintext CBD.

2.2 Background

2.2.1 Data Deduplication

Similar to data compression that identifies *intra-file* redundancy, deduplication is used to eliminate both *intra* and *inter* file duplicates. In general, chunk-based dedupe can capture "smaller" redundancy within files and thus often yields higher deduplication ratio $dr = \frac{originial\ dataset\ size}{stored\ dataset\ size}$, or the space reduction percentage $sr = 1 - 1/dr$ [41]. In storage backup scenario, the "original dataset" is an accumulated collection of all the data before deduplication from previous backup cycles. Further, if dedupe is allowed to be performed *cross users*, we usually can expect more space savings. On the other hand, dedupe occurring on the *server side* consumes more network bandwidth than the *client-side* dedupe, but with less privacy breach risk (see Section 2.4).

Chunking Algorithms

The data stream can be partitioned into *fixed-sized* chunks, which offers high processing rates and small computation overhead. However, it suffers from the boundary-shifting problem, where even a single bit added to the beginning of a file will result in different chunks [87]. A bit more CPU-intensive *variable-sized chunking* method can be used to address this problem. Briefly, this algorithm adopts a fixed-length sliding window to move onwards the data stream byte by byte. If the fingerprint (typically Rabin's fingerprint [20]), of the data segment covered by the window, satisfies a certain condition, this segment is marked as a partition point. The region between two consecutive partition points constitutes a chunk (see [78, 87] for detailed discussion). Variable-sized chunking provides users with more storage savings and is widely used in practice [87]. In addition, we can apply advanced *content-aware* chunking algorithms to identify duplicates on semantic information level [14, 71, 73], given the knowledge of file type, format, statistics information, etc. It turns out to be useful in speeding up the online chunk key generation (see Section 2.5.1).

Chunk Fragmentation

A succinct chunk ID is computed by applying a hash function, such as SHA1[1], over this chunk. We can determine whether the chunk has already been stored by looking up a key-value index table that maintains unique chunk IDs and their corresponding chunk storage locations. To achieve high write performance, each unique chunk is not directly written into the storage; instead, it is stored into a fixed-sized container (typically 2MB or 4MB) in the cache and the whole container is flushed to the storage once it is full. To read a chunk from storage, the entire container storing the chunk is retrieved. Therefore, it is likely that data restoration needs to read the shared chunks physically dispersed over different containers. A higher chunk fragmentation means more severe physical dispersion, which ends up with read performance degradation [61, 70]. We can use the average number r of *containers read per MB* to measure the fragmentation level and evaluate the read performance by *speed factor* $1/r$ [70].

2.2.2 Convergent Encryption

CE is extensively used in secure dedupe systems [10, 39, 40, 72, 90, 103, 119] as a prominent instantiation of message-locked encryption (MLE) [2, 11]. More precisely, to encrypt a file f with CE, we first locally derive the CE key $k = h(f)$, where h is a secure hash function, e.g., SHA256. Next, we use any secure symmetric encryption Enc, such as AES128, with secret key k to obtain the ciphertext $c = Enc(k, f)$. Apparently, the deterministic encryption process will always generate the same ciphertext c for the same plaintext f and enable ciphertext deduplication. It is worth noting that CE only provides security guarantees for unpredictable data and is inherently vulnerable to offline brute-force attack [10, 72]. In this work, we introduce a server-aided CE in the sense that the secret key is still derived from the target chunk but with the assistance of a dedicated key server (see Section 2.5).

2.2.3 Blind RSA Signature

In a server-client model, blind signature allows the server to cryptographically sign the secure hash of a message from the client without disclosing the message content. In a blind

[1]Note that the security vulnerability of SHA1 is orthogonal to its application here in dedupe setting.

RSA signature [27], let $\{N, e, d\}$ be a valid set of RSA parameters, where the modulus N is the product of two large primes p and q, $ed = 1 \bmod \varphi(N)$ and $\gcd(e, \varphi(N)) = 1$. $\varphi(N) = \text{lcm}(p-1, q-1)$. Then the public key is (e, N) and the private key is d.

- $z \leftarrow \mathsf{MessageMask}(msg, r)$: The client prepares a random number $r \in \mathbb{Z}_n$ and a full domain hash $\text{H} : \{0,1\}^* \to \mathbb{Z}_n$. He masks his original message msg by $z = \text{H}(msg) r^e \bmod N$;
- $\theta' \leftarrow \mathsf{Sign}(z)$: The server signs z with the private key and sends the signature $\theta' = z^d \bmod N$ back to the client;
- $\theta \leftarrow \mathsf{Unmask}(\theta')$: On the client side, the intended signature on msg is derived from $\theta = \theta' r^{-1} \bmod N$ and can be verified by $\text{H}(msg) \overset{?}{=} \theta^e \bmod N$.

In Section 2.5, we will show how to build the ROKG protocol on top of the blind RSA signature and further improve its efficiency.

2.3 Related Work

In the literature, there are in general two approaches, i.e. server-aided and serverless schemes, to prevent the direct key derivation from the data by the client.

2.3.1 Server-aided Encryption Solutions

Server-aided solutions (including ours) is more suitable for the enterprise/organization network and transparent to the established deduplication services. Puzio *et al.* [90] proposed to use an honest proxy server to encrypt the CE-generated ciphertexts by the client before uploading them to a storage server. Their scheme claims to provide secure chunk-level deduplication but it is unclear how to mitigate online brute-force attack in the malicious model. By the adoption of an identity server, Stanek *et al.* in [103] presented a secure file-based deduplication scheme that prevents online brute-force attack from masqueraded clients. However, only non-private popular files can be deduplicated by using a threshold encryption. Bellare *et al.* [10] proposed a server-aided secure dedupe system in the enterprise setting. By blind RSA signature, the CE key can be obliviously generated. The offline brute-force attack is prevented since the compromised storage server cannot access the key server. However, the online attack is still possible by controlling a legitimate client. As a result, all the client's storage can be revealed by the attack. They applied a per-client file-based rate-limiting method to slow down the online attack.

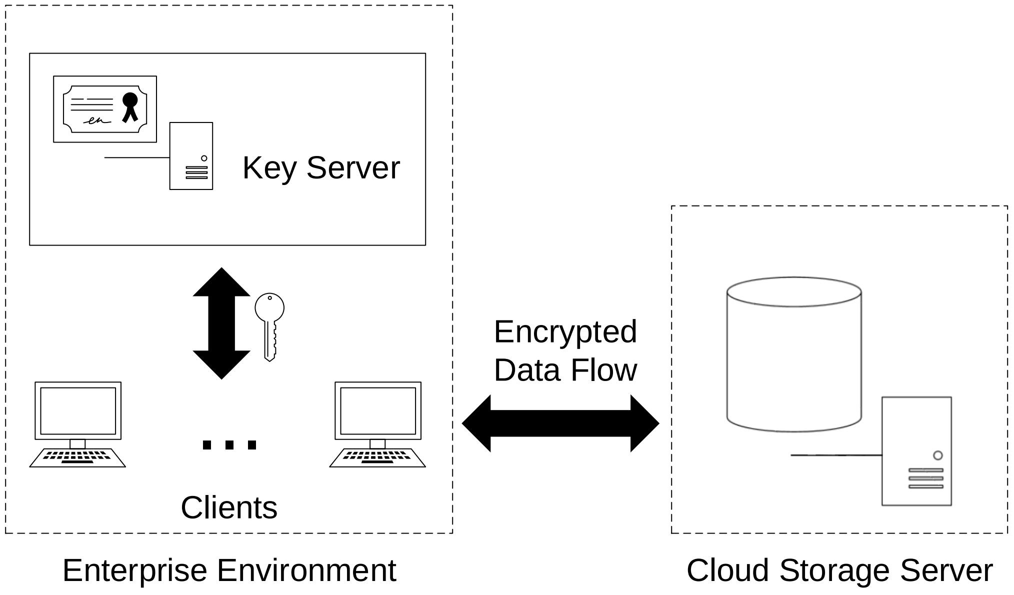


Figure 2.1: Framework of the proposed scheme.

2.3.2 Serverless Encryption Solutions

Duan [40] proposed to replace the role of a key server with clients using a modified Shoup RSA threshold signature scheme. It is unclear how to enforce any rate-limiting policy to slow down the brute-force attack. Xu *et al.* [119] proposed to deduplicate the message ciphertexts generated by randomized encryption. It only stores the first-uploaded file. For the same file uploading request, it provides the file encryption key to the user, which in turn is encrypted by the file. If the user indeed owns the file, he can derive the key and decrypt the file ciphertext. Brute-force attacks are avoided by assuming that the storage server is honest and cannot be compromised. In [72], the authors introduced a cross-user deduplication scheme. For an already stored file, a client executes a password-authenticated key exchange protocol with online clients who have previously uploaded the same file to obtain the CE key. Note that this process still needs to be coordinated by the storage server. Similar to [10], the offline brute-force attack is impossible because the CE key is not self-generated. They also adopt a per-file rate-limiting strategy to bound how many online protocol instances for each file can be invoked. Notice that these solutions cannot be directly integrated into the existing cloud storage services without substantial modification.

Remark 2.1. Besides failing the protection of low-entropy chunks, applying the above-mentioned schemes to chunk-based deduplication will incur a considerable performance penalty in key generation and deduplication.

2.3.3 Other Security Aspects

The cross-user client-side deduplication may introduce side-channel attacks. By uploading a crafted file to the storage server and observing the deduplication process, the adversary can learn additional information about the file, e.g. whether it has been uploaded by other clients, etc. This is more devastating for predictable data. Harnik *et al.* [56] proposed a randomized threshold approach to alleviate such side channel attack. To avoid private data leakage by using a single hash, Halevi *et al.* [55] proposed a proof-of-ownership (PoW) framework to verify the ownership of the file that the client is trying to access. Recently, Li *et al.* [69] presented a practical attack to reveal the deduplicated ciphertext storage by frequency analysis due to the deterministic nature of CE/MLE. Along another research line, the authors in [68] proposed a CE key management scheme that applies deduplication over encryption keys and distributes the key shares across multiple key servers. Chen *et al.* in [29] proposed an MLE scheme also with the focus on key management in the CBD setting. However, they did not consider the protection of low-entropy chunk and practical dedupe performance.

2.4 Problem Statement

2.4.1 System Model

There are three entities in our secure client-side cross-user deduplication system, key server ($\mathcal{KS}$), clients ($\mathcal{C}$'s) and public cloud storage server ($\mathcal{SS}$) as shown in Figure 2.1. We consider a periodical file backup service provided to $\mathcal{C}$'s in an enterprise network. The key server $\mathcal{KS}$ is set up in charge of client authentication and chunk encryption key generation. Specifically, a client $\mathcal{C}_j$ performs the chunking algorithm on his backup data. $\mathcal{KS}$ authenticates $\mathcal{C}_j$ upon request and generates the CE key k for each chunk ch of $\mathcal{C}_j$'s backup data in an oblivious manner. Then $\mathcal{C}_j$ encrypts the data chunks with the associated keys and uploads ciphertexts to $\mathcal{SS}$[2], such as Microsoft Azure Backup. $\mathcal{SS}$ stores the deduplicated incoming data stream in the corresponding containers before writing them into storage. As a result, the entire data protection phase, including key generation and data encryption, is transparent to $\mathcal{SS}$. $\mathcal{SS}$ only offers a basic and simple interface to its clients as in the plaintext data backup scenario.

[2]For simplicity, we omit the non-security steps, such as chunk ID generation and index table lookup on $\mathcal{SS}$.

2.4.2 Security Model

We focus on protecting the confidentiality of predictable data in this work because we can achieve semantic security for unpredictable data with CE. $\mathcal{KS}$ learns nothing about $\mathcal{C}_j$'s input chunk during the protocol execution. A compromised $\mathcal{SS}$ can launch *offline brute-force attack* by enumerating ciphertexts of predictable file candidates and comparing them with the target ciphertext in an offline manner. Although enterprise network is usually protected by enforcing rigorous security policies, we assume that it is possible for an external adversary to compromise a limited number of internal clients. Thus the adversary can perform an *online brute-force attack* by further accessing $\mathcal{KS}$.

We first define an ideal functionality $\mathcal{F}_{dedupe}$ of our scheme. The input of $\mathcal{F}_{dedupe}$:

- The client $\mathcal{C}_j$ has an input chunk ch;
- The key server $\mathcal{KS}$'s input is a chosen secret d_t;
- The cloud storage server $\mathcal{SS}$ has no input.

The output of $\mathcal{F}_{dedupe}$:

- $\mathcal{C}_j$ obtains the chunk key k;
- The output of $\mathcal{KS}$ is θ';
- $\mathcal{SS}$ gets the ciphertext $c = Enc(k, ch)$ and learns whether it has been stored.

We will prove our scheme secure in the malicious model if a probabilistic polynomial-time (PPT) adversary cannot distinguish the real-world execution of the proposed scheme and an ideal-world protocol that implements the functionality $\mathcal{F}_{dedupe}$ in the presence of a PPT simulator. In addition, we do not consider side-channel attacks, proof of ownership and key management in this study. Our system design will complement the current research [10, 29, 55, 56, 68, 69, 72]. Further, we assume that all the communication channels between $\mathcal{KS}$, $\mathcal{C}_j$ and $\mathcal{SS}$ are secure, and cannot be eavesdropped or tampered with by the adversary.

2.4.3 Design Goals

We devise a privacy-preserving chunk-based dedupe system aiming to achieve the following design goals. Pertaining to security,

- Realize the ideal functionality $\mathcal{F}_{dedupe}$ in the malicious model;
- Prevent offline brute-force attack by $\mathcal{SS}$;

- Mitigate online brute-force attack by slowing down its speed and providing multi-client compromise resilience.

In the performance aspect,

- Realize efficient chunk encryption key generation;
- Our design should be comparable with the plaintext CBD with respect to performance, such as dedupe ratio and data restore speed.

2.5 Protocol Design

In this section, we elaborate on our protocol design and provide discussion on the adopted techniques.

2.5.1 Randomized Oblivious Key Generation

Chunk encryption key can be generated by running a secure (oblivious) two-party computation between $\mathcal{C}_j$ and $\mathcal{KS}$, so that $\mathcal{KS}$ learns nothing on the $\mathcal{C}_j$'s input and algorithm output while $\mathcal{C}_j$ cannot infer $\mathcal{KS}$'s secret. In general, such desired protocol can be realized by any blind signature scheme. Here we use the widely-adopted blind RSA signature similar to prior work [10] and further introduce the randomness into the oblivious key generation.

Algorithm Definition

Let the hash functions $\mathrm{G} : \mathbb{Z}_n \rightarrow \{0,1\}^l$ and $\mathrm{H} : \{0,1\}^* \rightarrow \mathbb{Z}_n$. We define the ROKG algorithm as follows.

Definition 2.2. (ROKG algorithm) The proposed randomized oblivious key generation for a total of s clients in the system consists of four fundamental algorithms.

- $\mathsf{Setup}(\lambda_r, \lambda_n) \rightarrow (\{PK, MK\})$: The setup algorithm takes as input the security parameters λ_r and λ_n and outputs n sets of RSA parameters $\{(N_i, e_i, d_i) | 1 \leq i \leq n\}$. Thus, the public parameters are $PK = \{(N_i, e_i)\}$ and master secrets are $MK = \{d_i\}$.
- $\mathsf{ChObf}(ch, r, PK_i, \mathrm{H}) \rightarrow z$: This chunk obfuscation algorithm takes as input the chunk data ch, a random number r, the associated $PK_i = \{N_i, e_i\}$ and hash function H. It outputs the obfuscated chunk data z.

- $\mathsf{OKeyGen}(MK_i, z) \to \theta'$: This oblivious chunk key generation algorithm takes as input the associated master secret $MK_i = d_i$ for the client and obfuscated chunk z. It outputs the corresponding obfuscated chunk key θ'.
- $\mathsf{KeyRec}(\theta', \mathrm{H}, \mathrm{G}, PK_i, r) \to k$ or $\perp$: This chunk key recovery algorithm takes as input θ', hash functions G and H, the associated public parameter PK_i and the random number r. If θ' is successfully verified, it outputs the chunk encryption key k. Otherwise, it outputs $\perp$.

ROKG Construction

In what follows, we provide the concrete ROKG design.

System setup. At the setup phase, the key server $\mathcal{KS}$ calls the Setup algorithm to generate n pairs of $\{(PK_i, MK_i)\}$. PK is published to all the clients $\mathcal{C}$'s. MK is kept as the master secrets for the following protocol execution.

Client registration. Each new client $\mathcal{C}_j$ in the system needs to be authorized and registered by $\mathcal{KS}$ before he can request the chunk encryption key. Specifically, for the authorized $\mathcal{C}_j$, $\mathcal{KS}$ uniformly at random selects a master secret MK_i from MK and stores the tuple $(id(\mathcal{C}_j), i)$ on the user list. The selection i is then returned to $\mathcal{C}_j$.

Chunk data mask. For a chunk data ch, the client $\mathcal{C}_j$ calls the algorithm ChObf to obfuscate ch before sent to $\mathcal{KS}$. In particular, $\mathcal{C}_j$ chooses the corresponding $PK_i = \{N_i, e_i\}$ and a random number r. Then he masks the original chunk by $z = \mathrm{H}(ch)r^{e_i} \bmod N_i$ and sends z to $\mathcal{KS}$.

Obfuscated chunk key generation. Upon receiving the key generation request from $\mathcal{C}_j$, the key server prepares the corresponding MK_i and PK_i by looking up the user list. $\mathcal{KS}$ then calls the $\mathsf{OKeyGen}$ algorithm to generate the obfuscated chunk key $\theta' = z^{d_i} \bmod N_i$ and returns it to the client.

Key recovery. The client $\mathcal{C}_j$ invokes the algorithm KeyRec to derive the real chunk encryption key k. Specifically, he first unmasks θ' to $\theta = \theta' r^{-1} \bmod N_i$. $\mathcal{C}_j$ then verifies θ by $\mathrm{H}(ch) \overset{?}{=} \theta^{e_i} \bmod N_i$. If θ is valid, he can further recover the chunk encryption key $k = \mathrm{G}(\theta)$.

The proposed ROKG algorithm hides $\mathcal{C}_j$'s input chunk ch and actual output key k from $\mathcal{KS}$ while protecting $\mathcal{KS}$'s secrets MK from prying eyes of the client.

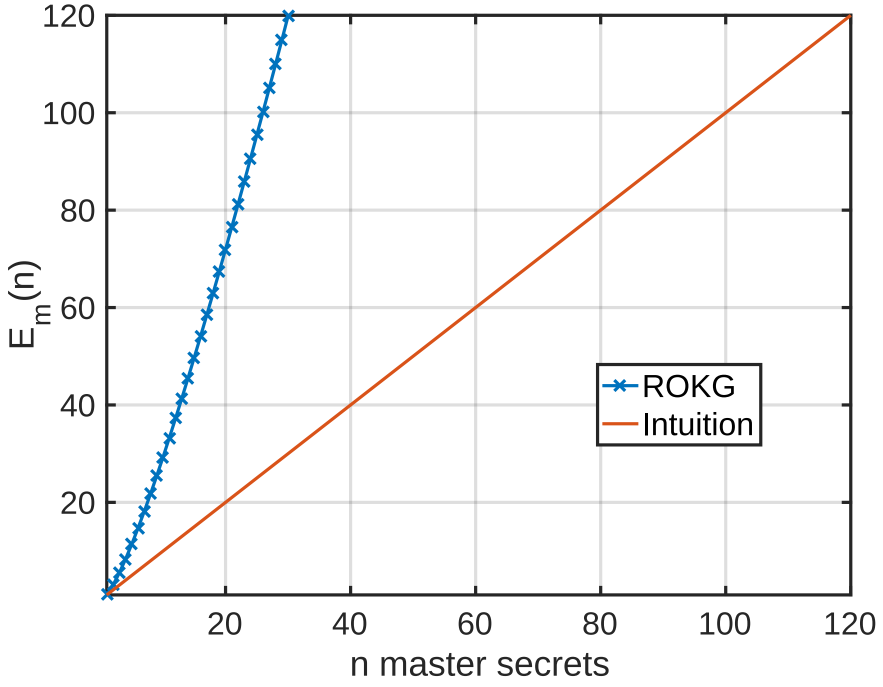


Figure 2.2: $E_m(n)$ with the increased number n of the master secrets in the system.

Asymmetry between Security Gain and Dedupability Loss

Intuitively, the security gain grows linearly with the increased number n of master secrets in the system but the dedupe effectiveness also degrades at the comparable rate. However, our proposed ROKG scheme brings us a counter-intuitive asymmetry property between the security gain and deduplication loss due to the characteristics of the accumulated storage backup. In other words, with the increased n, the growth rate of protection is much larger than that for dedupe performance loss. In what follows, we elaborate the impact of ROKG on these two aspects.

Multi-client compromise resilience. In prior work [10, 90, 103], once a legitimate client is compromised, the entire encrypted storage of all clients can be revealed using online brute-force attack by taking $\mathcal{KS}$ as a key oracle. In contrast, we introduce randomness into the key generation. Obviously, given any compromised $\mathcal{C}_j$ out of s clients in the system, the adversary can only infer at most $\frac{s}{n}$ clients' data on $\mathcal{SS}$ $(s \geq n)$. On the other hand, the adversary is able to increase his advantage by compromising more clients. In previous works [10, 72], compromising one client suffices for the whole storage exposure. Here we care about the equivalent situation of revealing the storage under all n secrets by controlling m clients and quantify the leakage.

We define our security gain as $E_m(n)$, which is the expected number of clients to be compromised for accessing all n secrets. $E_m(n)$ can be denoted by $n(1+\frac{1}{2}+\cdots+\frac{1}{n})$ from the insights of the coupon collector's problem [37]. By intuition, the growth rate of $E_m(n)$ is expected to be comparable with that of n. However, using ROKG gives us a much faster increase in $E_m(n)$ as shown in Figure 2.2. Therefore, we can choose a relatively small n but stay resilient to more compromised clients. We also provide the accurate probability $P_n(m)$ for the case that the adversary compromises m clients in order to infer the storage under all n master secrets of $\mathcal{KS}$ $(m \geq n)$. In general, there are n^m possible ways, in which we are interested in the number of functions from a set of m elements to a set of n elements. Such number can be denoted by $n!S(m,n)$. $S(m,n) = \frac{1}{n!}\sum_{j=0}^{n}(-1)^{n-j}\binom{n}{j}j^m$ is the Sterling number of the second kind [104]. Therefore, the probability is $P_n(m) = \frac{n!S(m,n)}{n^m}$. Given a fixed n, $P_n(m)$ grows as expected by compromising more clients shown in Figure 2.3. It also exhibits that the whole system becomes more robust under the attack by increasing n.

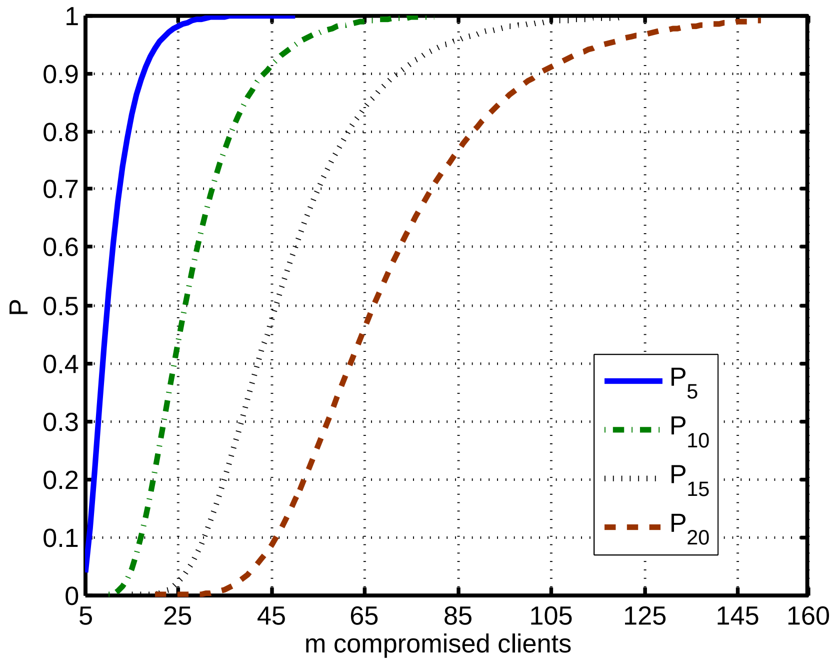


Figure 2.3: $P_n(m)$ with m compromised clients when $n = 5,\ 10,\ 15,\ 20$.

In practice, we can tweak the parameter n to accommodate the real network scale. The number of compromised machines in an enterprise network usually depends on not only the company's size but also its security policies/controls. The typical infection percentage ranges from 0.1% to 18.5% [35]. To demonstrate the effectiveness of our protocol, we take into account a fairly protected small company of 100 employees with 10% or lower infection ratio (10 compromised clients). Thus, for $n = 5$ the adversary will succeed only with the probability less than 50%. Note that we are free to adopt a larger n to further make $P_n(m)$ negligible.

Impact on deduplication. Indeed, the resulted threat isolation comes at the price of dedupe effectiveness loss. This is two folds. First, we study the case for one backup cycle. W.l.o.g, the stored data can be represented by $x + y$ under one secret. x is the size of data that cannot be deduplicated across all the users. y refers to the size of data that have been deduplicated. We consider the best case that all the users share the same data portion of y. Thus, $x + y$ is the lower bound of stored data size we can achieve in reality. By using n secrets in the system, the size of the stored data is $x + ny$. Compared to the dedupe ratio dr_1 under one key, the dedupe ratio using n keys is $dr_n = \frac{x+y}{x+ny} \cdot dr_1$. Obviously, the performance degradation does not follow the simple linearity, which is also demonstrated by our experiment (see Section 2.7). If x outsizes y significantly, selecting a small or moderate n will not introduce an obvious performance penalty. The example may be that the backup contains data types not naturally suited for deduplication, such as compressed files commonly seen in the archive storage, the rich media data (e.g. videos, images). Therefore, our scheme can provide better security guarantee in this situation. Otherwise, non-negligible dedupe loss is expected for this one-time backup scenario. On the other hand, the periodic backup service will eventually give rise to a high dedupe performance with our scheme. This is because the size of the accumulated backup storage is a more dominant factor in the dedupe ratio computation compared to the orders of magnitude smaller n. Thus, we can take advantage of this asymmetry to achieve stronger privacy protection with a larger n while enjoying comparable space savings with the plaintext CBD. This analysis is consistent with our experiment (see Section 2.7). In addition, our scheme enables better read performance (see Section 2.5.3).

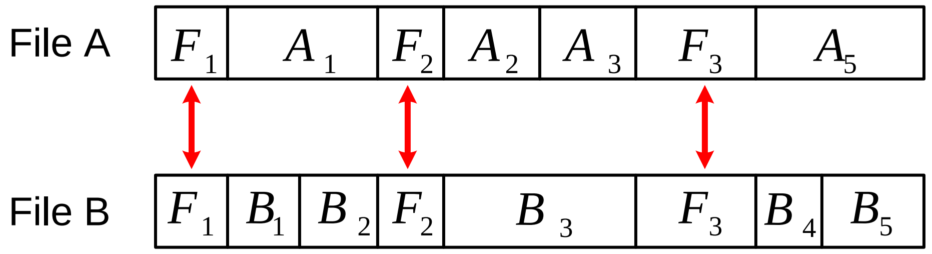


Figure 2.4: F_1, F_2, and F_3 are immutable parts identified by content-aware chunking algorithms in two file copies A and B of the same file format.

Efficiency Improvement for Frequent Insensitive Data

We observe that files sharing the similar contents or with the same data type, e.g., .pdf, .doc, may contain identical data fields. Intuitively, if we extract these immutable parts and utilize them as a file fingerprint, we can accelerate the key generation significantly. In this case, we modify the original ROKG protocol for the frequent insensitive data as follows.

The setup and client registration remain the same. $\mathcal{C}_j$ first adopts the content-aware deduplication [14, 71, 73] to identify the common data parts for his files with the same format or data type. For instance, in Figure 2.4, given the extracted common data chunks F_1, F_2, and F_3, we can compute the file format fingerprint $h_f = \mathrm{H}(F_1||F_2||F_3)$. Instead of running the remaining algorithms, i.e. ChObf, OkeyGen, and KeyRec for each chunk, $\mathcal{C}_j$ uses the fingerprint h_f as the input to get the file format key k_f. Subsequently, $\mathcal{C}_j$ produces the chunk key $k = \mathrm{G}(k_f||ch)$ offline for all the chunks ch in the file with the same fingerprint. Therefore, we have a constant computation and communication overhead for the modified ROKG protocol. For files containing sensitive information, $\mathcal{C}_j$ still needs to run the online protocol per chunk with $\mathcal{KS}$ to stay more resilient to the brute-force attack.

2.5.2 Slowing down Online Brute-force Attack

Online brute-force attack can be launched by compromising a legitimate client and interacting with the key server to obtain the chunk encryption key. Completely preventing such attack is still an open problem. Using our ROKG design only partially mitigate this issue. On the other hand, rate-limiting strategy is broadly used to *slow down* this online attack in file-based dedupe scenario [10, 72]. We propose to enforce a *per-backup* rate-limiting policy in the chunk-based dedupe system, which is inspired by the observed features of storage backup in practice. Specifically, given the projected backup data size and expected chunk size, we set a budget

$$q = \frac{\textit{projected backup data size}}{\textit{expected chunk size}}$$

for each client to bound the number of requests that are allowed to be processed by $\mathcal{KS}$ during the prescribed time window, e.g. 2:00 – 3:00 AM every Tuesday. Otherwise, $\mathcal{KS}$ will not respond to the client.

This policy is made based on the following observations. First, the enterprise backup workloads usually exhibit periodicity, i.e., they follow the scheduled time window and update cycle. Moreover, it is expected that the content and size of the periodical backup data from an enterprise user does not change rapidly [78]. For example, a weekly 2GB OS snapshot of a client's machine is backed up to the cloud storage with the expected chunk size 8KB. We can estimate a weekly backup budget $q = 250,000$ for each client. We may also set an additional buffer to tolerate the error and ensure the success of the backup. We assume that any attempt to use the budget for the attack without actually storing the data, or only storing a portion below the budget will be detected in a post auditing process. In addition, our approach is supposed to work with both full and incremental backup (only storing deltas between files) scenarios. Note that the rate-limiting strategy may not be fully compatible with the proposed content-aware key generation mechanism because the adversary can circumvent the online restriction by offline computation. Thus, it is desired to enforce the policy for sensitive data.

2.5.3 Improving Data Restore Speed

There are several reasons why we are concerned about read performance even in the backup storage. First of all, data restore speed is considered critical for crash/corruption recovery, where higher read speed results in shorter recovery window time. Furthermore, we need to reconstruct the original data stream (more frequent than user-triggered data retrieval) for staging the backup data streams to archive storage in light of limited capacity of deduplication storage [79].

Despite the reduced dedupe ratio, the proposed scheme will naturally enable better read performance for a user as we allow a duplicate chunk copy under one secret to be kept in the storage without referring it to an existing copy under another secret in an old container. As a result, we trade off deduplication for faster user backup restore speed, which happens to reflect a similar optimization philosophy in plaintext dedupe research [14, 79, 101]. We can further improve read performance by adopting a reconstruction-aware chunk placement mechanism to enforce a high spatial locality for chunks. Specifically, the system maintains a set of dedicated chunk containers $cnt_{i,j}$ in the cache for each $\mathcal{KS}$ secret d_i, where $1 \leq i \leq n$ and j indicates a distinct container for the same key[3]. We achieve high spatial locality by storing $cnt_{i,j}$ in separate locations of the disk according to i, such as in different partitions. Therefore, chunks under the same $\mathcal{KS}$ secret are stored close to each other. When restoring a client's data, read access is only restricted to a limited scope of the disk instead of random accessing the whole storage.

We argue that the proposed chunk placement will not disclose the additional information except what has been learned by the adversary. In particular, while improving the read performance, the adversary may identify clients under the same $\mathcal{KS}$ secret by observing their chunks stored in the same set of containers $cnt_{t,j}$. However, such information leakage is inevitable in any dedupe system, which the adversary on $\mathcal{SS}$ always knows from deduplication process. Furthermore, we can leverage the encrypted data search techniques to realize the secure chunk retrieval and verification [109] or combine ORAM to hide the access pattern [49], which are interesting future research directions.

2.6 Security Analysis

In this section, we show that the presented scheme accomplishes our security goals. As we discussed, our proposal alleviates the online brute-force attack even when multiple clients are compromised and we can diminish the attack efficiency by the *per-backup* rate-limiting strategy. In what follows, we show that $\mathcal{F}_{dedupe}$ and offline brute-force attack prevention are achievable as well.

[3]We need an additional 1-byte field *cnt_num* in the chunk metadata to mark which set of containers this chunk should go to.

Theorem 2.3. *Our secure chunk-based deduplication protocol computing the ideal functionality $\mathcal{F}_{dedupe}$ is secure in the malicious model if the blind RSA signature is secure in the random oracle model and the hash function G is modeled as a random oracle.*

Proof. (Sketch) Assume the blind RSA signature protocol to be an oracle that takes in parties' inputs and then sends outputs to them. For simplicity, we only consider one $\mathcal{KS}$ secret in the system, whose security can be easily extended to the multi-secret situation. Suppose that there is a simulator $\mathcal{S}$ for the corrupted parties in the ideal world. $\mathcal{S}$ can access $\mathcal{F}_{dedupe}$ in the ideal world and record message transcript from the protocol execution in the real world. Therefore, the adversary $\mathcal{A}$ cannot distinguish the view $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ constructed by $\mathcal{S}$ in the ideal world from the view $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$ in the real-world protocol execution.

- Corrupted $\mathcal{C}_j$: Assume that $\mathcal{SS}$ and $\mathcal{KS}$ are honest in this case. Simulator $\mathcal{S}$ records the calls $\mathcal{C}_j$ makes to the blind RSA signature with the input chunk ch. It invokes the ideal functionality $\mathcal{F}_{dedupe}$ with the same ch. $\mathcal{S}$ also records the output θ from the blind RSA signature and keeps a list $\{(\theta, k)\}$. If $\mathcal{S}$ receives a θ that appears on the list, it returns the corresponding k as the chunk key. Otherwise, k is a random number and $\mathcal{S}$ writes it back onto the list. In the end, $\mathcal{F}_{dedupe}$ also outputs a chunk key k_Δ for ch and θ'_Δ. $\mathcal{S}$ may simulate the message transcript as follows. It sets $r = \frac{\theta'_\Delta}{\theta}$. z_Δ is simulated as $r^e\theta^e$ and $h_\Delta = \theta^e$. If $\mathcal{C}_j$ behaves honestly, k is equal to k_Δ. On the other hand, $\mathcal{A}$ can deviate from the protocol by modifying his inputs and replacing elements that are sent to $\mathcal{S}$. In this case, θ and k are random numbers in light of the RSA signature and random oracle G. $\mathcal{S}$ can simulate the message transcript similar to the above. The message transcripts cannot be computationally distinguished by adversary $\mathcal{A}$ in the real and ideal worlds. Thus, $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ and $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$ are identically distributed.

- Corrupted $\mathcal{KS}$: Assume that $\mathcal{C}_j$ and $\mathcal{SS}$ are honest. $\mathcal{F}_{dedupe}$ outputs θ'_Δ for $\mathcal{KS}$. $\mathcal{S}$ can simulate the incoming message z_Δ similarly to the above. $\mathcal{KS}$ can deviate from the designated protocol execution by replacing the signature, which, however, will only pass the client-side verification with negligible probability given the unforgeability of blind RSA signature. Thus, $\mathcal{A}$ still cannot distinguish $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ and $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$.

- Corrupted $\mathcal{SS}$: Assume that both $\mathcal{C}_j$ and $\mathcal{KS}$ are honest. By providing $\mathcal{F}_{dedupe}$ with the input ch and d_i, $\mathcal{S}$ receives the chunk ciphertext $c_\Delta = Enc(k_\Delta, ch)$. If ch exists, the ideal-world c_Δ is the same as c in the real world. Otherwise, the chunk key and ciphertext are random in both the real and ideal worlds. $\mathcal{SS}$ can deviate from the protocol by modifying the ciphertext, which may result in a failed chunk deduplication. Thus, $\mathcal{A}$ cannot distinguish $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda_r)$ and $\mathbf{Real}_{\mathcal{A}}(\lambda_r)$.

□

According to Theorem 2.3, the proposed protocol securely computes the ideal functionality $\mathcal{F}_{dedupe}$. In addition, the adversary (in the case of corrupted $\mathcal{SS}$) cannot generate the encryption key from chunk data itself or access $\mathcal{KS}$, thereby preventing the offline brute-force attack.

2.7 Performance Evaluation

We implement our secure chunk-based deduplication system on the real-world backup storage from File systems and Storage Lab at Stony Brook University [44]. We focus on the 2013 MacOS Snapshots dataset collected on a Mac OS X Snow Leopard server with 54 users. There are 249 snapshots (daily backup) in total with the duration of 11 months, and each snapshot is generated by using variable-sized chunking with an average chunk size of 8KB. To simulate our enterprise backup setting, we synthesize each individual user's daily backup by extracting his files from the snapshot and incorporate data of `UID-0` as the base file system. The total size of the backup storage we considered here before deduplication is roughly 463 TB. We also set the size of the chunk container and LRU cache as 4 MB and 512 MB respectively. We do not use the relevant optimization technique, such as parallelization. The corresponding experimental results are an average of 100 trials.

The existing secure chunk-based designs [29, 68, 90], with the different research focus, do not consider the protection of low-entropy data in the presence of a strong attacker and the practical deduplication performance, such as fragmentation level. Their performance should be roughly the same as that of plaintext CBD in terms of dedupe ratio and data restore speed because they heuristically keep unique chunk copy in the system. We will not explicitly mention them hereafter and only compare the proposed scheme with plaintext CBD instead (see Sections 2.7.2 and 2.7.3).

2.7.1 Online Key Generation

We use Python to implement our TCP-based randomized oblivious key generation protocol between the key server $\mathcal{KS}$ and client $\mathcal{C}_j$. The server machine is equipped with a 3.1 GHz AMD FX 8120 processor and 32GB DDR3 memory. On the same LAN, the client machine has an Intel i3-2120 processor with 12GB memory. The blind RSA signature is implemented with RSA1024 and SHA256. CE is instantiated by CTR[AES128].

Latency

We measure the latency incurred by the proposed online protocol, which is defined as the time between that $\mathcal{C}_j$ sends out the request to and receives the response from $\mathcal{KS}$. As shown in

Figure 2.5(a), the overhead of the basic scheme without optimizing the chunk key generation is linear in the size of the client's data. A large size backup data stream produces more chunks, which will invoke more per-chunk key generation protocol instances. Figure 2.5(a) also shows that the protocol latency is dramatically reduced and becomes constant if we resort to the content-aware chunking and only run the protocol to generate the format key k_f. Thus, the latency merely depends on the number of distinct file formats in the backup stream of the client (supposing all are predictable files), regardless of the size.

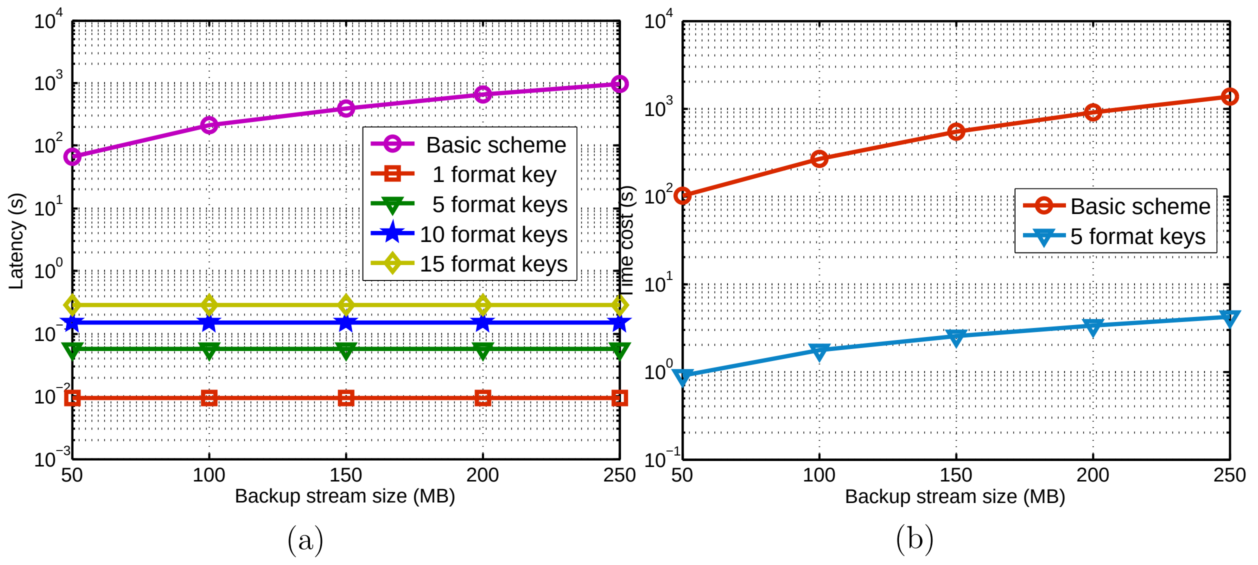


Figure 2.5: (a) Latency and (b) Client-side computation overhead for the online key generation protocol with increased size of backup stream of a client. We invoke an online protocol instance for each chunk in our basic scheme.

Client-side Cost

We also measure the additional client-side cost due to the ROKG protocol and convergent encryption. As shown in Figure 2.5(b), the overhead is mainly contingent on the size of the backup data stream. It is worth noting that this cost still demonstrates the linearity with the total backup data size even using the proposed efficient key generation algorithm. This is because client still needs to carry out the offline hashing for each chunk key, and encrypts them individually. Again, it shows a significant performance advantage by the efficient key generation protocol.

2.7.2 Deduplication Effectiveness

We develop a deduplication simulator with C code to measure the corresponding dedupe performance of the proposed scheme and compare our protocol with plaintext CBD.

Adhering to our previous analysis (see Section 2.5.1), for a single backup attempt, for example, the $40th$ backup in Figure 2.6, the dedupe ratio dr_5 with 5 $\mathcal{KS}$ secrets is about half of dr_1 in the plaintext CBD, which exhibits a non-linear performance loss with n. Figure 2.6 also shows that the space reduction percentage sr is sensitive to the number of backups and increases as periodic backup service continues. With more backup date added, sr gradually approaches the plaintext dedupe performance regardless of the number of master secrets used in the system because the size of the accumulated "original dataset" dominates the computation for sr. As a result, we can adopt more $\mathcal{KS}$ secrets to achieve stronger privacy protection with a minimal dedupability loss.

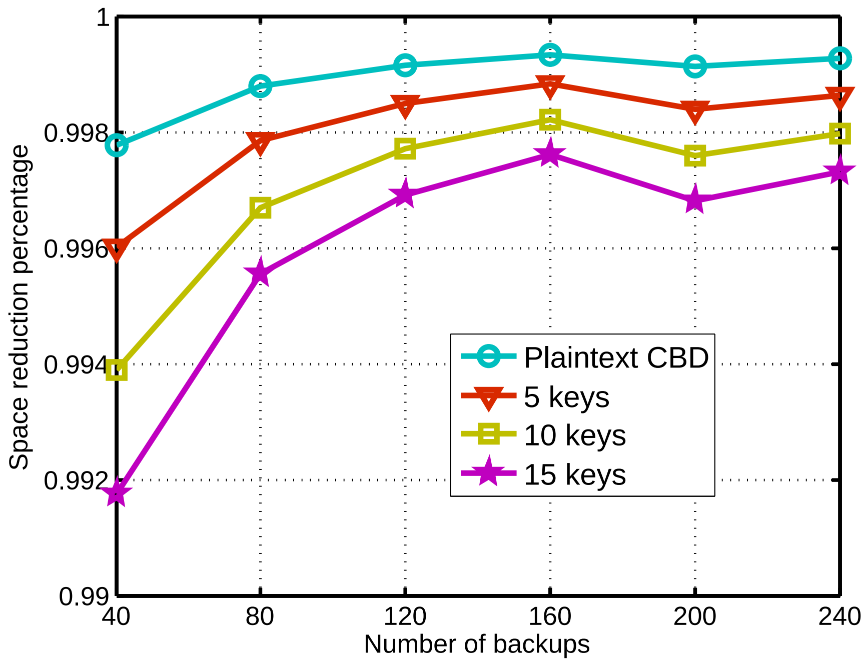


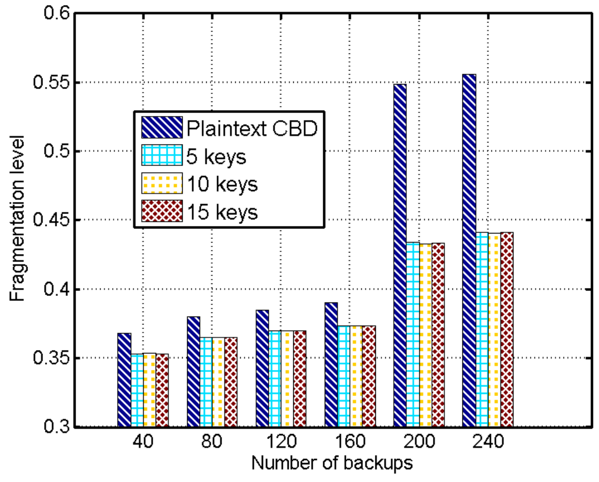
Figure 2.6: Space savings sr with the increased number of backup storage in the cloud. Each backup includes snapshots of 54 users' machines. The comparison is drawn between the plaintext CBD, and our scheme using 5, 10, and 15 $\mathcal{KS}$ secret keys.

2.7.3 Fragmentation

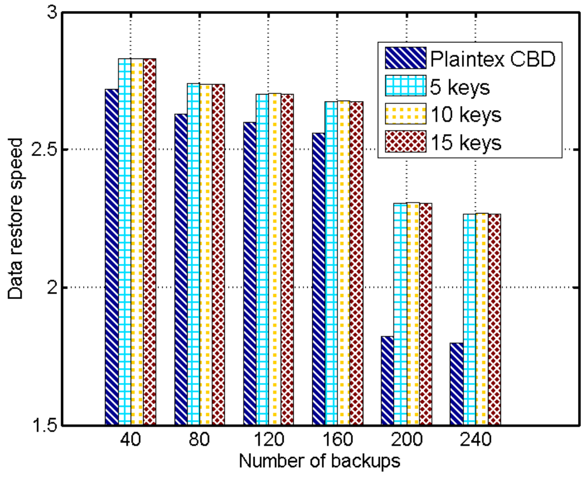
We focus on the fragmentation/read performance comparison with plaintext CBD because of the sequentially written/read files and defragmentation schedule in file-based deduplication. Although our design shows a slight loss of dedupability so as to achieve the desired security objectives, the chunk fragmentation level r for user backup is also reduced, thereby the increased data restore speed shown in Figure 2.7(a) and Figure 2.7(b) respectively.

We neglect the actual low-level data placement on disk (i.e. our high spatial locality design in Section 2.5.3), and other common optimization techniques (e.g. large container/cache).

We use the same simulator to study the impact of fragmentation on a user's backup data caused by our security design only. The result, as shown in Figure 2.7(a), again validates the importance of the fragmentation issue in the chunk-based deduplication that the more data added to the storage, the more severe the chunk fragmentation level. Figure 2.7(a) also shows that our design can maintain a lower fragmentation level than plaintext CBD. In addition, the number of $\mathcal{KS}$ secret keys used in the system has a negligible impact on the chunk fragmentation. Note that we can enjoy substantial fragmentation reduction with the increased size of backup storage. Figure 2.7(b) accordingly shows that the proposed scheme achieves better user backup read performance than plaintext CBD. As a result, we can use more $\mathcal{KS}$ secret keys to realize faster restore speed and stronger privacy guarantees at the same time.



(a)



(b)

Figure 2.7: (a) Fragmentation level r and (b) data restore speed $1/r$ with the increased number of backups in the cloud. Both fragmentation level and read performance are measured by recovering the backup dataset of a randomly selected user. All the comparisons are drawn between the plaintext CBD, and our scheme using 5, 10, and 15 $\mathcal{KS}$ secret keys.

2.8 Summary

We in this chapter discuss and address challenges in designing a data deduplication system at the chunk level. The impact of online brute-force attack can be substantially alleviated by allowing clients to invoke the proposed randomized oblivious key generation protocol with a key server and enforcing a per-backup rate-limiting policy. We also exploit the content-aware deduplication technique to further improve the efficiency of the online key generation. Our scheme is on par with the plaintext practice in terms of the deduplication performance while gaining better security guarantees compared to the existing work.

Chapter 3

Efficient Verifiable Conjunctive Keyword Search over Large Dynamic Encrypted Cloud Data

3.1 Introduction

While cloud computing provides unparalleled benefits to its users in a "pay-as-you-go" manner, such as on-demand computing resource configuration, ubiquitous and flexible access, considerable capital expenditure savings, etc., security concern is still the major inhibitor of cloud adoption for many large companies, organizations and individuals [114]. Encrypting sensitive data before uploading them to cloud storage, e.g., Google Drive, Dropbox, etc., can avoid user privacy breach, but the obfuscated data thwart the cloud to quickly sort out intended information as per user-selected keywords of interest.

In the literature, encrypted data search is proposed to address the above challenges, but the majority of these schemes [23, 24, 34, 50, 62, 63, 106] assume that the cloud server is *semi-trusted*. In other words, the server will not deviate from the designated protocol and return erroneous search result. This assumption is usually insufficient in the real world due to the underlying software/hardware malfunctions, financial incentives (cloud server may intentionally save computational resources and return false result), or even the existence of a *malicious* server controlled by an outside attacker, etc. Therefore, cloud users may desire a more trustworthy secure search system beyond the *semi-trusted* model, i.e., they can be assured of the authenticity of returned search result in a more challenging scenario where a fully *malicious* cloud server exists. Furthermore, the result verification cost should be minimal and affordable to users irrespective of the outsourced large data collection. Otherwise it will not be of practical value considering the dramatically increasing number of resource-constrained mobile devices.

On the other hand, a preferred *verifiable* search scheme should be constructed without sacrificing other critical search functionalities. One of these is *conjunctive keyword search* [23, 24, 50, 106], i.e., it allows the cloud server to produce search result containing all the queried keywords within one search operation. This multi-keyword search capability not only boosts search efficiency, but also improves the overall user experience. Moreover, a practical scheme should also work for *dynamic data* [62, 63, 67, 105], i.e., search can be conducted even after *inserting*, *deleting*, or *modifying* a file, which is specially appealing to users who would like to update their files while retaining the encrypted data search functionality without rebuilding the whole system from scratch.

In this chpater, aiming to provide all the above search functionalities in a challenging *malicious* model while preserving search privacy, we propose an efficient verifiable conjunctive keyword search scheme (VCKS) over large dynamic encrypted cloud data. We use the *inverted index* structure [34, 66, 67] to build our secure index and allow data user to delegate her search task to a cloud server. We exploit the *bilinear-map accumulator* [6, 36] technique to construct an authenticated data structure. As such the user can verify the returned search result either *privately* by herself or with the assistance of a *public* TA. The proposed VCKS scheme also supports file collection update, i.e., *insertion*, *deletion* and *modification*. Finally, the extensive experimental evaluation shows the efficiency and practicality of our scheme.

3.2 Related Work

Secure search technique has been achieved in both symmetric [23, 24, 34, 50, 62, 63, 106] and asymmetric [16, 108, 115] settings with a variety of search functionalities investigated in the literature.

3.2.1 Static Search

In the symmetric setting, Curtmola *et al.* [34] proposed an efficient secure single-keyword search scheme, and gave a formal security notion, i.e., *security against chosen-keyword attack* (CKA1) and a stronger notion of *adaptive security against chosen-keyword attack* (CKA2). To enrich the search functionality, secure multi-keyword search was realized in [24, 50] (conjunctive keyword search) and [23, 106] (conjunctive and disjunctive keyword search). Furthermore, Sun *et al.* [106] improved the search efficiency and accuracy using a tree-based index structure and the *cosine similarity measure* in the *vector space model*. In the public key scenario, Boneh *et al.* [16] presented the first public key encryption with keyword search scheme constructed from identity-based encryption. Recently, Sun *et al.* [108] proposed the first attribute-based keyword search scheme to realize fine-grained owner-enforced search authorization. Note that the above schemes only support *static data*, and are secure against a *semi-trusted* server.

3.2.2 Dynamic Search

Goh [48] proposed a dynamic secure search scheme but the bloom filter based index may introduce false positive into the final search result. Chang *et al.* [26] also presented a dynamic search solution with linear search time. Kamara *et al.* [63] proposed a dynamic version of [34], supporting data insertion and deletion on the outsourced dataset, and proved it CKA2-secure. Later they accelerated the search process by using parallelization technique [62]. However, these works *will not* be secure against a *malicious* adversary, and users cannot verify the authenticity of returned search result.

Table 3.1: Comparison of verifiable search solutions.

Scheme	Query type	Dynamism	Verifiability	PPE[1]
[116]	single	static	private	no
[74]	range	dynamic	private	no
[66]	single	static	private	no
[67]	single	dynamic	public/private	no
[107]	conjunctive[2]	static	private	no
[110]	conjunctive	static	private	no
[105]	single	dynamic	private	no
This work	conjunctive	dynamic	public/private	yes

[1] PPE = Practical performance evaluation.
[2] This work also supports disjunctive keyword search.

3.2.3 Verifiable Search

Wang *et al.* [116] use the hash chain to verify the single keyword search result. In [74], a verifiable logarithmic-time search scheme was presented to support range queries. Kurosawa *et al.* proposed the first UC-secure verifiable search scheme with single keyword [66] and extended it to a dynamic version [67] later. For static data, Sun *et al.* proposed the first verifiable multi-keyword (conjunctive and disjunctive) search with hash and signature techniques in [107] and later presented a verifiable attribute-based keyword search in [110]. Stefanov *et al.* [105] recently gave a dynamic encrypted data search scheme with small search privacy leakage, which enables result verification for single keyword search. It is worth noting that most of these search verification mechanisms are heuristic constructions without evaluating the practical performance, especially for large-scale dataset stored in the cloud. In addition, no scheme can achieve *conjunctive*, *dynamic*, and *publicly/privately verifiable* search at the same time as shown in Table 3.1.

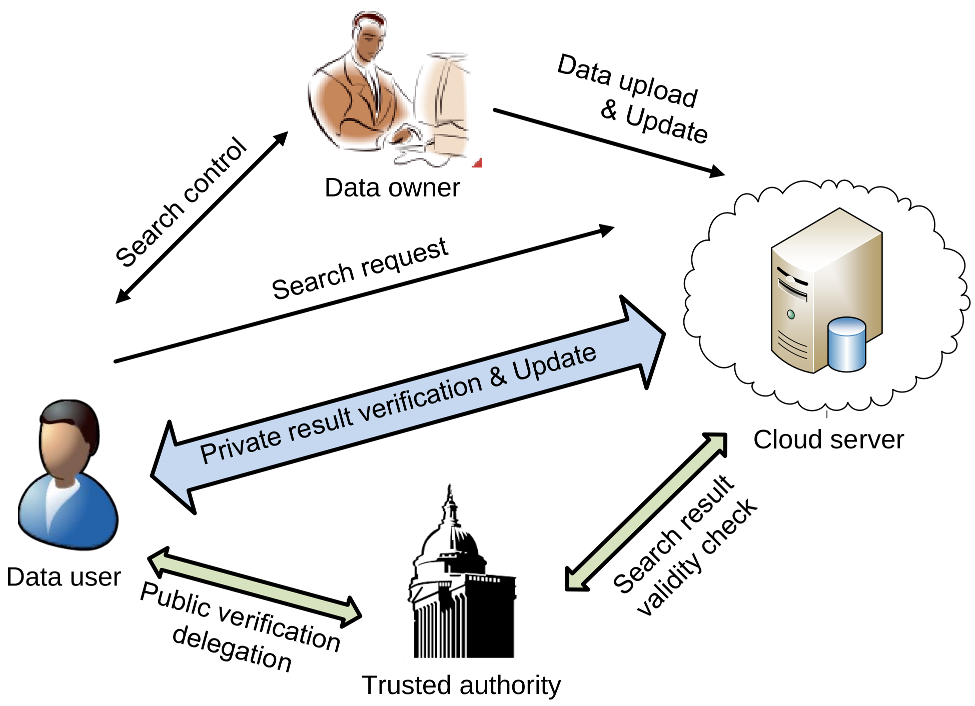


Figure 3.1: System model.

3.3 Problem Formulation

Our proposed VCKS scheme consists of three main entities: *data owner*, *data user*, and *cloud server*, as shown in Figure 3.1. Data owner first prepares ciphertexts $C = \{c_1, c_2, ..., c_n\}$ for the file collection $F = \{f_1, f_2, ..., f_n\}$ of size n by using any secure encryption algorithm, such as AES. She also generates an encrypted index with a pre-defined dictionary $\mathcal{W} = \{w_1, w_2, ...w_m\}$ containing m keywords and verification related data for these files. Then data owner uploads all the above information to cloud server. Later she can update the server-hosted file collection arbitrarily, i.e., file *insertion*, *deletion* or *modification*. Authorized data users are able to obtain a search token from data owner for multiple keywords of interest and other auxiliary information via the search control mechanism [34], which is outside the scope of this work. On receiving the search token from data user, server performs the *conjunctive keyword search* over the secure index of C. Our scheme supports both *private* and *public* search result verification as shown in Figure 3.1. For the latter, data user can offload the computational burden of verification to a *public* TA. In this case, server returns the result and its proof to the TA. The TA will send the result to the user if it is valid. Otherwise, it notifies the user of its rejection.

3.3.1 Definition of VCKS

We give the definition of our scheme in the following.

Definition 3.1. *(Verifiable conjunctive keyword search).* A verifiable conjunctive keyword search scheme for dynamic data is a tuple (Setup, Enc, $\mathsf{GenTree}$, $\mathsf{GenToken}$, Search, $\mathsf{GenProof}$, $\mathsf{UpdToken}$, Update, Verify, Dec) of ten polynomial-time algorithms such that:

- $(K, s, pub) \leftarrow \mathsf{Setup}(1^\lambda)$: On input a security parameter λ, this probabilistic algorithm outputs secret keys K, s, and other public parameters pub.

- $(\gamma, C) \leftarrow \mathsf{Enc}(K, \delta, F)$: On input a secret key K, an index δ and a set of files F, this probabilistic algorithm outputs an encrypted index γ, and a set of ciphertexts C.

- $T \leftarrow \mathsf{GenTree}(s, \delta, C)$: On input a secret key s, an index δ and ciphertexts C, this deterministic algorithm outputs an accumulation tree T (see Section 3.4).

- $\tau_Q \leftarrow \mathsf{GenToken}(K, Q)$: On input K and an intended keyword set $Q = \{w_{j_1}, w_{j_2}, ..., w_{j_t}\} \subseteq \mathcal{W}$, this (possibly probabilistic) algorithm outputs a search token τ_Q.

- $C(Q) \leftarrow \mathsf{Search}(\gamma, \tau_Q, C)$: On input an encrypted index γ, a search token τ_Q and ciphertexts C, this deterministic algorithm outputs the search result $C(Q)$ for the file list L_Q, where each ciphertext c_i contains all the intended keywords in Q and its identifier i is included in L_Q.

- $\Pi \leftarrow \mathsf{GenProof}(C(Q), T, pub)$: On input a search result $C(Q)$, an accumulation tree T for the file storage and public parameters pub, this deterministic algorithm outputs a proof Π.

- $\tau_u \leftarrow \mathsf{UpdToken}(u, f_i, d(v))$: On input an update operation $u \in \{\textit{modify, insert, delete}\}$, a file f_i and corresponding digests $d(v)$ for nodes v in T, this (possibly probabilistic) algorithm outputs an update token τ_u.

- $(\gamma', C', T') \leftarrow \mathsf{Update}(\gamma, C, T, \tau_u)$: On input an encrypted index γ, a set of ciphertexts C, an accumulation tree T and an update token τ_u, this deterministic algorithm outputs new γ', C', and T'.

- $(accept, reject) \leftarrow \mathsf{Verify}(C(Q), \Pi, d(r), pub)$: On input a search result $C(Q)$, a proof Π, a root digest $d(r)$ from data owner, and parameters pub (also the secret key s in the case of private verification), this deterministic algorithm outputs *accept* if the search result is valid; else, it outputs $reject$.

- $f \leftarrow \mathsf{Dec}(K, c)$: On input K and a file ciphertext c, this deterministic algorithm outputs a plaintext file f.

3.3.2 Security Definition

Privacy

Almost all the existing secure search schemes [23, 34, 50, 62, 63, 66, 67, 105, 106] leak *search pattern*, i.e., whether the same keyword was used for search in the past or not, and *access pattern*, i.e., after searching keywords in Q, the file list L_Q is disclosed. In practice, these privacy information cannot be preserved efficiently. Thus, we in this work do not aim to protect them. Similar to [62], we define two stateful leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ to precisely capture what is being revealed by ciphertext and the tokens: 1) $\mathcal{L}_1(\delta, F)$. On input the index δ and the file collection F, this function outputs the dictionary size $|\mathcal{W}|$, the file collection size $|F|$, the file identifiers i and its size $|i|$, and the size of each file $|f_i|$. For update, this function also reveals the identifiers and/or the size of the corresponding files; 2) $\mathcal{L}_2(\delta, F, Q)$. Given the index δ, the file collection F, and the keyword set Q searched in the past, this leakage function reveals *search* and *access patterns*.

We adapt the *privacy* definition in [67] to a dynamic conjunctive keyword search setting, where a *semi-trusted* server is considered. Note that this security definition is slightly stronger than CKA2 security defined in [34, 62, 63].

Definition 3.2. *(Privacy).* For a dynamic conjunctive keyword search scheme as given in Definition 3.1, we consider the following experiments, where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator, and $\mathcal{L}_1$ and $\mathcal{L}_2$ are stateful leakage functions.

$\mathbf{Real}_{\mathcal{A}}(\lambda)$: The challenger runs $\mathsf{Gen}(1^\lambda)$ to generate a key K. $\mathcal{A}$ sends a tuple (F, δ) to the challenger and receives $(\gamma, C) \leftarrow \mathsf{Enc}(K, \delta, F)$. The adversary makes a polynomial number of queries by picking $q \in \{Q, (u, i)\}$. If $q = Q$ is a search query then the adversary receives a search token $\tau_Q \leftarrow \mathsf{GenToken}(K, Q)$ from the challenger. If $q = (u, i)$ the adversary receives from the challenger an update token $\tau_u \leftarrow \mathsf{UpdToken}(u, i)$. Finally, $\mathcal{A}$ outputs a bit b.

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}$ chooses a tuple (F, δ). Given $\mathcal{L}_1(\delta, F)$, simulator $\mathcal{S}$ outputs a tuple (γ, C) and returns it to $\mathcal{A}$. In the search phase, the adversary makes a polynomial number of queries by picking $q \in \{Q, (u, i)\}$. If $q = Q$ is a search query, reveal $\mathcal{L}_2(\delta, F, Q)$ to $\mathcal{S}$ and return τ_Q generated by $\mathcal{S}$ to $\mathcal{A}$. If $q = (u, i)$, $\mathcal{S}$ is given the updated output of $\mathcal{L}_2(\delta, F, \{w_j\})$ and sends τ_u to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit b in this experiment.

We say that our dynamic conjunctive keyword search scheme in Definition 3.1 satisfies *privacy* if there exists a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, $|Pr[\mathbf{Real}_{\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]|$ is negligible.

Verifiability

Due to possible data corruption, software/hardware malfunctions, and even the existence of a *malicious* server in the system, search result returned to the user may be false or contain errors. The data user should be able to detect such misbehavior to guarantee the validity of the search operation. Specifically, given a valid search result $C(Q)$ and its proof Π for a search token τ_Q , the adversary $\mathcal{A}$ wins if she can forge invalid $C^*(Q)$ and Π^* that will pass the Verify algorithm. We have the following definition.

Definition 3.3. *(Verifiability).* A verifiable and dynamic conjunctive keyword search scheme in Definition 3.1 satisfies *verifiability* if for any PPT adversary $\mathcal{A}$, the probability of successfully forging search result and its proof is negligible for any fixed $(F, \mathcal{W}, \gamma)$ and search tokens τ_Q.

UC-Security

The security of a protocol proven in a stand-alone setting is preserved under composition if it is secure in the universally composable security framework [22]. In the UC framework, an environment $\mathcal{Z}$ exists to produce all the input and read all the output in the system, and arbitrarily interacts with an adversary $\mathcal{A}$. We say a protocol securely realizes a given functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can tell wether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal world. We define the ideal functionality $\mathcal{F}$ of our proposed VCKS scheme in what follows.

Definition 3.4. *(Ideal functionality $\mathcal{F}$).* The adversary $\mathcal{S}$ is only given $\mathcal{L}_1(\delta, F)$ and $\mathcal{L}_2(\delta, F, Q)$ in this ideal world. The ideal functionality $\mathcal{F}$ interacts with user (data owner or data user) P_1, server P_2 and adversary $\mathcal{S}$, and runs as below:

- On receiving (F, δ) from P_1, verify that it is the first upload input from P_1. If so, store (F, δ), and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$. Otherwise discard this input.
- On receiving search token τ_Q from P_1, reveal $\mathcal{L}_2(\delta, F, Q)$ to $\mathcal{S}$. If $\mathcal{S}$ returns "accept", send search result to P_1; else, send "reject" to P_1.
- On receiving update token τ_{mod} from P_1, replace corresponding file in F and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$.
- On receiving update token τ_{del} from P_1, delete corresponding file in F and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$.
- On receiving update token τ_{in} from P_1, insert corresponding file to F and reveal $\mathcal{L}_1(\delta, F)$ to $\mathcal{S}$.

3.4 Preliminaries

3.4.1 Bilinear-map Accumulator

Bilinear-map accumulator [6, 36] is an efficient data authentication mechanism that provides a constant-size digest for an arbitrarily large set of inputs, and a constant-size witness for any element in the set such that it can be used to verify the (non-)membership of the element in this set. Bilinear-map accumulator can be realized using either *symmetric* or *asymmetric* pairing. For ease of illustration, we adopt the symmetric version in this study.

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic multiplicative group with the same prime order p. g is a generator of $\mathbb{G}$. Thus, a bilinear pairing is defined as $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the properties of *bilinearity, computability* and *non-degeneracy*. To construct a bilinear-map accumulator, we generate an accumulation value $acc(L) = g^{\prod_{a_i \in L}(a_i+s)}$ in $\mathbb{G}$ for a set L of n elements $\{a_1, a_2, ..., a_n\}$ in $\mathbb{Z}_p^*$, where $s \in \mathbb{Z}_p^*$ is a randomly chosen value and $\prod_{a_i \in L}(a_i + s)$ is a *characteristic polynomial* for the set L. For any subset $L' \subseteq L$, a witness $Wit_{L',L} = g^{\prod_{a_i \in L-L'}(a_i+s)}$ can be produced. Subsequently, the subset test can be carried out by checking

$$e(g^{\prod_{a_i \in L'}(a_i+s)}, Wit_{L',L}) \overset{?}{=} e(acc(L), g). \tag{3.1}$$

Note that only given corresponding elements a and $\{g^{s^i} : 0 \leq i \leq q\}$ where q is an upper bound on n, $g^{\prod(a+s)}$ can be constructed with polynomial interpolation [89]. The security of the bilinear-map accumulator is derived from the q-strong bilinear Diffie-Hellman (q-SBDH) assumption [86].

This data structure can also support update operation. For example, to insert a new element a_{n+1} into the set L, we can obtain a new set accumulation value $acc'(L) = acc(L)^{(a_{n+1}+s)}$, and $acc'(L) = acc(L)^{(a_i+s)^{-1}}$ is an updated accumulation value after deleting some element a_i from L.

3.4.2 Accumulation Tree

To support efficient integrity check over multiple sets in one data structure, we extend bilinear-map accumulator to a *collision-resistant* accumulation tree [86]. Specifically, suppose there are m sets $\{L_1, ..., L_j, ..., L_m\}$, for each of which $acc(L_j)$ is computed. By choosing a constant $0 \leq \epsilon \leq 1$, a tree T can be generated with $l = \lceil 1/\epsilon \rceil$ levels and m leaves. Each leaf node v represents a particular set L_j in the set collection. It stores the accumulation value $acc(L_j)$ and $d(v) = acc(L_j)^{(s+j)}$ (this proves that L_j refers to $acc(L_j)$). Each internal node v of this constant-height tree T has degree $O(m^\epsilon)$ and contains the hash $d(v)$ of a set of its children $N(v)$, where $d(v) = g^{\prod_{u \in N(v)}(s+h(d(u)))}$ and $h : \mathbb{G} \to \mathbb{Z}_p^*$ is a collision-resistant hash function. Hence, the integrity of the set is protected by its accumulation value and the accumulation tree protects the integrity of all the accumulation values stored in the leaves. For instance, Figure 3.2 shows a 2-degree accumulation tree of 2 levels for sets L_1, L_2, L_3 and L_4 by selecting $\epsilon = 0.5$.

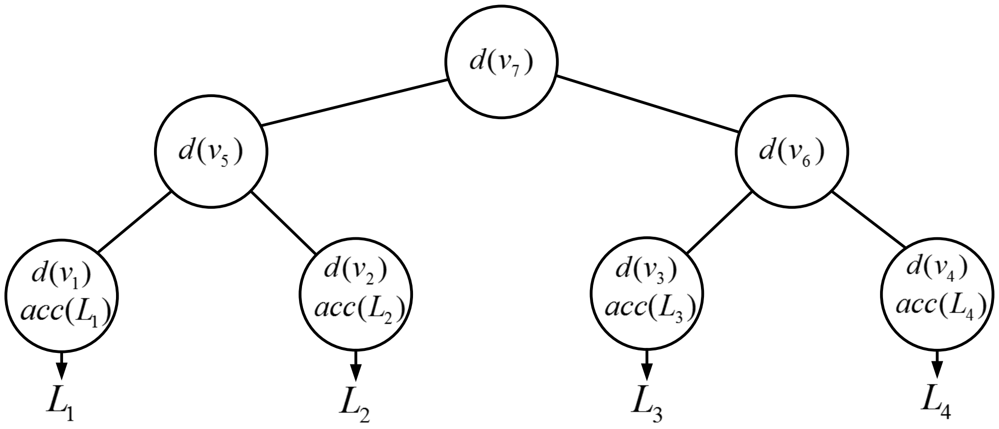


Figure 3.2: Example of an accumulation tree with $\epsilon = 0.5$.

Not only does an accumulation tree support update operation on dynamic data collection, which is inherent from bilinear-map accumulator, it also can be used to verify set operations, such as set intersection. More precisely, given t queried sets $\{L_{j_1}, L_{j_2}, ..., L_{j_t}\}$, the intersection set $I = L_{j_1} \cap L_{j_2} \cap ... \cap L_{j_t}$ should satisfy the following two conditions.

- **Subset**: $I \subseteq L_{j_1} \cap I \subseteq L_{j_2} \cap ... \cap I \subseteq L_{j_t}$;
- **Completeness**: $(L_{j_1} - I) \cap (L_{j_2} - I) \cap ... \cap (L_{j_t} - I) = \emptyset$.

To meet the first requirement, the verifier only needs to check Equation 3.1. As for completeness condition, suppose $A_{j_b}(s)$ is the characteristic polynomial of set $L_{j_b} - I$ for $1 \leq b \leq t$. We need find another t polynomials $P_{j_b}(s)$ such that $\sum_{b=1}^{t} P_{j_b}(s)A_{j_b}(s) = 1$, which can be computed efficiently by Euclidean algorithm. Thus we obtain the completeness witnesses $Cwit_{I,L_{j_b}} = g^{P_{j_b}(s)}$ accordingly. Given the subset witnesses $Wit_{I,L_{j_b}} = g^{A_{j_b}(s)}$, we say the completeness condition is satisfied if the following equation holds:

$$\prod_{b=1}^{t} e(Cwit_{I,L_{j_b}}, Wit_{I,L_{j_b}}) \overset{?}{=} e(g,g). \tag{3.2}$$

3.5 Our Construction

By indexing the dataset using inverted index structure [67], we design our VCKS scheme with an efficient result verification mechanism that can be realized in both *public* and *private* settings. The index $\delta = \{a_{j,i}\}$ in our scheme is an $m \times n$ matrix as shown in Figure 3.3 such that if f_i contains the keyword w_j, then $a_{j,i} = 1$; otherwise set $a_{j,i} = 0$. We denote δ_j as the j^{th} row of δ. In what follows, we begin to describe our proposed VCKS scheme in terms of system level operations, i.e., **Data Upload**, **Search**, **Data Download**, **Update**, where each operation may contain one or more algorithms in Definition 3.1.

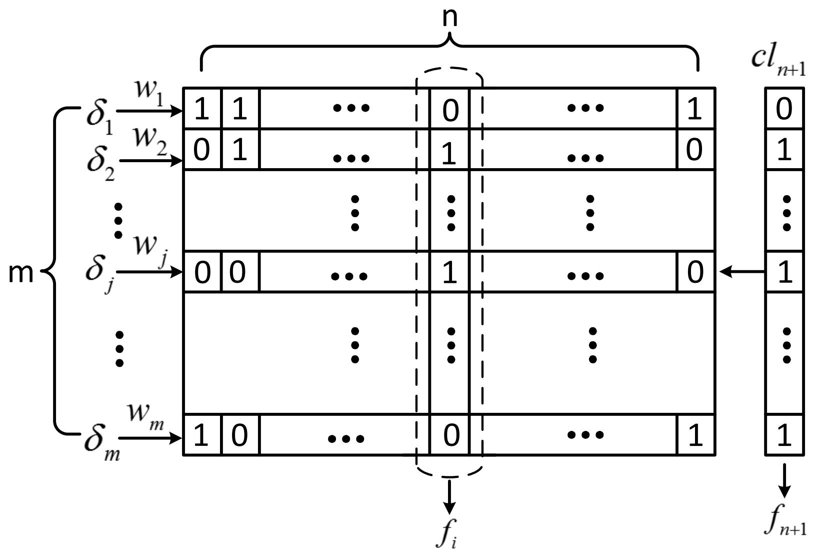


Figure 3.3: Illustration for matrix index δ and insertion operation for f_{n+1}.

3.5.1 Data Upload

In this initial operation, the data owner generates a secret key set $K = (k_1, k_2, k_3)$ by calling the algorithm Setup, where k_1 and k_2 are keys of a pseudorandom function prf_k, and the key k_3 is for the secure symmetric-key encryption algorithm Enc and decryption algorithm Dec shared with data user. The data owner uses the algorithm Enc to encrypt the file collection F into a ciphertext set C. She then prepares the secure index $\gamma = \{(\pi_{\varphi(j)}, \delta'_{\varphi(j)})_{1\leq j\leq m}\}$ as follows:

1. For each keyword $w_j \in \mathcal{W}$, compute $\pi_j = prf_{k_1}(w_j)$;

2. Set δ'_j equal to the first n bits of $\delta_j \oplus prf_{k_2}(w_j)$;

3. Apply a random permutation φ on $\{1, ..., m\}$.

The Setup algorithm also outputs a secret key s, and public parameters $pub = \{p, \mathbb{G}, \mathbb{G}_T, g, e, h, g^s, g^{s^2}, ..., g^{s^q}\}$, where $q = max\{m, n\}$. Subsequently, the data owner generates an accumulation tree T for index δ using the algorithm GenTree. For a leaf node v of T pointing to a ciphertext set $C(i)_j = \{c_i | a_{j,i} = 1\}$ associated with δ_j, compute digest $d(v) = acc(C(i)_j)^{(\pi_j+s)} =$

$g^{\prod_{c_i \in C(i)_j}(h(i,c_i)+s)(\pi_j+s)}$ and store it in this leaf node. Otherwise, let $d(v) = g^{\prod_{u \in N(v)}(h(d(u))+s)}$. $d(r)$ is the digest on the root node r of T.

The data owner retains the secret key s and all the node digests $d(v)$ for $v \in T$. Then she uploads the file ciphertexts C and the secure index γ along with the accumulation tree T to the cloud server.

3.5.2 Search

For a set Q of t intended keywords $\{w_{j_1}, ..., w_{j_t}\}$ from data user, the data owner calls the algorithm GenToken to obtain the search token $\tau_Q = \{(\alpha_{j_b} = prf_{k_1}(w_{j_b}), \beta_{j_b} = [prf_{k_2}(w_{j_b})]_{1...n})\}$ for $1 \leq b \leq t$, where β_{j_b} is the first n bits of $prf_{k_2}(w_{j_b})$, and returns it to the user.

After receiving the search token from the data user, the Search algorithm is invoked by the cloud server. More precisely, it identifies each tuple $(\pi_{\varphi(j_b)}, \delta'_{\varphi(j_b)})$ in the secure index γ if $\pi_{\varphi(j_b)} = \alpha_{j_b}$. Next, the cloud server is able to recover $\delta_{j_b} = \delta'_{\varphi(j_b)} \oplus \beta_{j_b}$ for $1 \leq b \leq t$. Finally, the search result $C(Q)$ can be derived by performing intersection operation on sets $\{\delta_{j_1}, \delta_{j_2}, ..., \delta_{j_t}\}$, where $c_i \in C(Q)$ contains all t keywords of interest in Q.

The server also prepares the proof Π for (public) result verification with the GenProof algorithm as below:

- Accumulation value $acc(C(i)_{j_b})$ and Π_{j_b} for each index row δ_{j_b}. Let $v_0, v_1, ..., v_l$ be the path in T from the leaf node v_0 associated with $acc(C(i)_{j_b})$ to the root node $v_l = r$. Set $\psi_z = g^{\prod_{u \in N(v_z) \setminus v_{z-1}}(h(d(u))+s)}$ for $z = 1, 2, ..., l$. As such, Π_{j_b} is defined as $\{(d(v_0), \psi_1), (d(v_1), \psi_2), ..., (d(v_{l-1}), \psi_l)\}$;

- Subset witness $Wit_{C(Q),C(i)_{j_b}}$ and completeness witness $CWit_{C(Q),C(i)_{j_b}}$, for $b = 1, 2, ..., t$;

- Coefficients $\sigma_0, \sigma_1, ..., \sigma_\rho$ of the characteristic polynomial for $\{h(i, c_i)\}$[1], where $c_i \in C(Q)$ and ρ is the size of the search result;

- The root node digest $d(r)$.

The cloud server returns all the encrypted files $C(Q)$ identified in L_Q and the proof Π.

[1]Given the roots of the polynomial, we can compute its coefficients efficiently by polynomial interpolation [89]. Accumulation value, subset witness and completeness witness can also be constructed by using the corresponding coefficients and public parameters $g, g^s, ..., g^{s^q}$.

3.5.3 Data Download

The data user first verify the search result in either *public* or *private* setting.

Input: Search result $C(Q)$, proof Π, root node digest $d(r)$ from data user and system parameters pub.
Output: "accept" or "reject".

```
1  Check $d(r)$ in $\Pi$ with that from data user, and the correctness of coefficients $\sigma_0, \sigma_1, ..., \sigma_\rho$.
   If any one fails output "reject", otherwise continue;
2  for $b = 1 \rightarrow t$ do
3      Check $e(d(v_0), g) \overset{?}{=} e(acc(C(i)_{j_b}), g^{\pi_{j_b}} g^s)$ (3.3);
4      for $z = 1 \rightarrow l-1$ do
5          Check $e(d(v_z), g) \overset{?}{=} e(\psi_z, g^{h(d(v_{z-1}))} g^s)$ (3.4);
6      end
7      Check $e(d(r), g) \overset{?}{=} e(\psi_l, g^{h(d(v_{l-1}))} g^s)$ (3.5);
8      If any one of the Equations 3.3, 3.4 and 3.5 fails, output "reject", otherwise continue;
9  end
10 Check subset condition by Equation 3.6. If it fails, output "reject", otherwise continue;
11 Check completeness condition by Equation 3.2. If it fails, output "reject", otherwise
   continue;
12 If none of the above fails, output "accept";
```

Algorithm 3.1: Public Search Result Verification

Public verifiability

The data user delegates the verification task to a *public* TA. In this scenario, the cloud server returns the search result and its proof to the TA. With only access to public parameters, i.e., g^{s^i}, the TA calls the algorithm Verify to verify the search result as illustrated in Algorithm 3.1. Note that the user also needs to send the latest root node digest $d(r)$ acquired from the data owner to the TA in order to facilitate the result verification (line 1)[2]. Given search result $C(Q)$, the coefficients can be verified efficiently (line 1) [86]. By checking Equations 3.3, 3.4 and 3.5 (line 3, 5 and 7 respectively), we can guarantee that the index row δ_{j_b} is associated with the $j_b{}^{th}$ leaf node of T. To check the subset condition for all the corresponding $C(i)_{j_b}$ in a batch manner (line 10), we make use of the equation below

$$e(\prod_{\iota=0}^{\rho}(g^{s^\iota})^{\sigma_\iota}, \prod_{b=1}^{t} Wit_{C(Q),C(i)_{j_b}}) \overset{?}{=} e(\prod_{b=1}^{t} acc(C(i)_{j_b}), g), \tag{3.6}$$

[2]The data owner can also sign this digest with a time stamp to guarantee the freshness of the search result, but the TA (or user in the private setting) needs additional cryptographic operations to verify this signature.

where g^{s^i} is from the public parameters *pub*. If the algorithm outputs "accept", $C(Q)$ is indeed the search result with respect to the queried keyword set Q. the TA will send it to the data user. The user then decrypts c_i to f_i by calling the algorithm Dec. Otherwise, the TA notifies the user of the rejection.

Input: Search result $C(Q)$, proof Π (exclusive of the coefficients $\sigma_0, \sigma_1, ..., \sigma_\rho$), root node digest $d(r)$ from data owner and system parameters *pub*.

Output: "accept" or "reject".

```
1  Check $d(r)$ in $\Pi$ with that from data owner. If it fails, output "reject", otherwise continue;
2  for $b = 1 \to t$ do
3      Check $e(d(v_0), g) \overset{?}{=} e(acc(C(i)_{j_b}), g^{(\pi_{j_b}+s)})$ (3.7);
4      for $z = 1 \to l-1$ do
5          Check $e(d(v_z), g) \overset{?}{=} e(\psi_z, g^{(h(d(v_{z-1}))+s)})$ (3.8)
6      end
7      Check $e(d(r), g) \overset{?}{=} e(\psi_l, g^{(h(d(v_{l-1}))+s)})$ (3.9);
8      If any one of the Equations 3.7, 3.8 and 3.9 fails, output "reject", otherwise continue;
9  end
10 Check subset condition by Equation 3.10. If it fails, output "reject", otherwise continue;
11 Check completeness condition by Equation 3.2. If it fails, output "reject", otherwise
     continue;
12 If none of the above fails, output "accept";
```

Algorithm 3.2: Private Search Result Verification

Private Verifiability

In case the TA is unreachable or does not even exist, we are able to achieve more computationally efficient *private* search verification by giving the secret key s to legitimate users as shown in Algorithm 3.2. The cloud server directly returns the result and its proof to the user. The proof Π does not include the coefficients $\sigma_0, \sigma_1, ..., \sigma_\rho$. Note that with secret key s, Equations 3.7, 3.8, 3.9 and 3.10 can be computed more efficiently than their counterparts in the *public* Verify algorithm. The subset condition can be verified by the following equation:

$$e(g^{\prod_{c_i \in C(Q)}(h(i,c_i)+s)}, \prod_{b=1}^{t} Wit_{C(Q),C(i)_{j_b}}) \overset{?}{=} e(\prod_{b=1}^{t} acc(C(i)_{j_b}), g). \tag{3.10}$$

3.5.4 Update

In a dynamic cloud storage, the data owner is able to *modify*, *insert* or *delete* files arbitrarily.

Modification

This update operation only results in a modified version f_i' of the original file f_i and has the file identifier i unchanged. Suppose that f_i' has the same keywords with f_i. Hence, the data owner does not need to update the secure index. By the algorithm $\mathsf{UpdToken}$, the data owner acquires the update token $\tau_{mod} = (i, c_i', \{d'(v)\})$. c_i' is the ciphertext of f_i' and $\{d'(v)\}$ is the modified digest set computed as follows. For the path $v_0, v_1, ..., v_l$ from a leaf node v_0 containing c_i' to root node $v_l = r$, update $d'(v_0) = d(v_0)^{(h(i,c_i)+s)^{-1}(h(i,c_i')+s)}$ and set $d'(v_z) = d(v_z)^{(h(d(v_{z-1}))+s)^{-1}(h(d'(v_{z-1}))+s)}$ for $z = 1, ..., l$. On receiving this update request from the owner, the cloud server updates the ciphertext set and accumulation tree.

Deletion

To delete a file f_i from the storage, the data owner adopts the $\mathsf{UpdToken}$ algorithm to generate an update token $\tau_{del} = (i, \{d'(v)\})$. The deletion operation is analogue to file modification except that for each leaf node containing c_i, set corresponding $d'(v_0) = d(v_0)^{(h(i,c_i)+s)^{-1}}$. Finally the server uses the Update algorithm to delete the original ciphertext c_i, and produce a new accumulation tree T'.

Insertion

To insert a new file f_{n+1} into current file collection, the algorithm $\mathsf{UpdToken}$ generates a new $(n+1)^{th}$ column cl_{n+1} for the matrix index δ as shown in Figure 3.3. For $1 \le j \le m$, $a_{j,n+1} = 1$ if the file contains keyword w_j; let $a_{j,n+1} = 0$ otherwise. Then cl_{n+1} is obfuscated to cl_{n+1}' by $a_{j,n+1} \oplus [prf_{k_2}(w_j)]_{n+1}$ and apply the random permutation φ to cl_{n+1}', where $[prf_{k_2}(w_j)]_{n+1}$ is the $(n+1)^{th}$ bit of $prf_{k_2}(w_j)$. The owner encrypts f_{n+1} to c_{n+1} by Enc. With the related new leaf node digests $d'(v_0) = d(v_0)^{(h(n+1,c_{n+1})+s)}$, an updated digest set $\{d'(v)\}$ can be computed. The corresponding update token τ_{in} is a tuple $(n+1, c_{n+1}, cl_{n+1}', \{d'(v)\})$, which allows the cloud server to update the file collection, the secure index and accumulation tree by calling the Update algorithm.

Remark 3.5. Data owner may keep a set of succinct file stubs $h(i, c_i)$ after **Data Upload** operation for update efficiency. The storage overhead is negligible compared with the size of F. Otherwise, she need sign them and interact with server every time the Update algorithm is triggered. In the *private* setting, data user with secret key s and file digest information from data owner can also update the file collection. This is a desirable feature in the case that the outsourced dataset is allowed to be written by multiple group users.

3.6 Security Analysis

In this section, we analyze the security properties of our proposed scheme and show that it achieves the defined security goals. We first prove that our VCKS scheme satisfies *privacy* in Definition 3.2 with a *semi-trusted* server (adversary). After incorporating the *verifiability* in Definition 3.3, our final scheme achieves the stronger notion of security, namely *UC-security* against a *malicious* adversary (see Section 3.3.2).

Theorem 3.6. *The VCKS scheme satisfies* privacy *in Definition 3.2.*

Proof. Let $\mathcal{A}$ and $\mathcal{S}$ be an adversary and a simulator in $\textbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ in Definition 3.2, respectively. Given the leakage function $\mathcal{L}_1(\delta, F)$, $\mathcal{S}$ outputs (γ', C') as follows. It simulates the encrypted file $c_i = \mathsf{Enc}_{k_1}(0^{|f_i|})$ for $i = 1, ..., n$, where k_1 is randomly selected for the CPA-secure encryption algorithm Enc, and $|f_i|$ is revealed by $\mathcal{L}_1$. To simulate the secure index $\gamma = \{(\pi_{\varphi(j)}, \delta'_{\varphi(j)})\}$ for $j = 1, ..., m$, $\mathcal{S}$ sets π_j as a random number and chooses $\delta_j \in \{0, 1\}^n$ at random. $\mathcal{S}$ then applies a random permutation φ on $\{1, ..., m\}$ and sends (C', γ') to $\mathcal{A}$.

Adversary $\mathcal{A}$ can make a polynomial number of queries by picking $q \in \{Q, (u, i)\}$. If q is a search query for a keyword set Q of t conjunctive keywords $\{w_{j_b}\}_{b=1,...,t}$, the leakage function $\mathcal{L}_2(\delta, F, Q)$ reveals L_Q to $\mathcal{S}$. Given this, for $b = 1, ..., t$ $\mathcal{S}$ can generate $a_{j_b,i} = 1$ if $i \in L_Q$; otherwise, set $a_{j_b,i} = 0$. Then $\mathcal{S}$ sets $\alpha_{j_b} = \pi_{\varphi(j_b)}$ and computes $\beta_{j_b} = \delta'_{\varphi(j_b)} \oplus (a_{j_b,1}, ..., a_{j_b,n})$. She returns $\tau'_Q = \{(\alpha_{j_b}, \beta_{j_b})\}$ to $\mathcal{A}$. If $q = (u, i)$ is an update query: 1) $u = modify$. Given $|f'_i|$ from leakage function, $\mathcal{S}$ simulates $c'_i = Enc_{k_1}(0^{|f'_i|})$. Then $\mathcal{S}$ sends $\tau'_{mod} = (i, c'_i)$ to $\mathcal{A}$; 2) $u = delete$. $\mathcal{S}$ returns $\tau'_{del} = i$ to $\mathcal{A}$; 3) $u = insert$. With $|f_{n+1}|$ from $\mathcal{L}_1(\delta, F)$, $\mathcal{S}$ computes $c_{n+1} = Enc_{k_1}(0^{|f_{n+1}|})$. Choose $cl'_{n+1} \in \{0, 1\}^m$ and apply the random permutation φ on it. Then $\mathcal{S}$ sends $\tau'_{in} = (n + 1, c_{n+1}, cl'_{n+1})$ to $\mathcal{A}$.

The adversary $\mathcal{A}$ cannot distinguish C' from C in experiment $\textbf{Real}_{\mathcal{A}}(\lambda)$ since Enc is CPA-secure. Due to the pseudorandom function prf used in $\textbf{Real}_{\mathcal{A}}(\lambda)$ $\mathcal{A}$ cannot distinguish γ' from γ either. Likewise, $\mathcal{A}$ cannot tell the differences between $\{\tau'_Q, \tau'_{mod}, \tau'_{del}, \tau'_{in}\}$ in $\textbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ and $\{\tau_Q, \tau_{mod}, \tau_{del}, \tau_{in}\}$ in $\textbf{Real}_{\mathcal{A}}(\lambda)$ because of the CPA-secure Enc, pseudorandom function prf and random permutation φ. Thus, $\mathcal{A}$ cannot distinguish $\textbf{Real}_{\mathcal{A}}(\lambda)$ and $\textbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$. □

Next, we prove the VCKS scheme secure against a *malicious* adversary.

Theorem 3.7. *The proposed VCKS scheme satisfies* privacy *in Definition 3.2 and* verifiability *in Definition 3.3.*

Proof. (Sketch) Similar to the proof of Theorem 1, we can prove *privacy* property. Therefore we only prove *verifiability.*

Suppose the adversary $\mathcal{A}$ can break *verifiability* with non-negligible probability for any fixed $(F, \mathcal{W}, \gamma)$ and search tokens τ_Q. $\mathcal{A}$ can produce $(C^*(Q), \Pi^*) \neq (C(Q), \Pi)$ but the *public* Verify algorithm outputs *accept*. $C(Q)$ and Π are valid search result and its proof respectively.

We will show that the probability of the above situation is negligible given the collision-resistant hash function h and the security of the accumulation tree proved in [86].

$C^*(Q) = C(Q)$ and $\Pi^* \neq \Pi$. If only $d^*(r) \neq d(r)$, the Verify algorithm will definitely output *reject*. If the coefficients in Π^* are computed incorrectly, the coefficient validity check will fail with high probability. If either the accumulation value or the related Π_j is incorrect, one of the Equations 3.3, 3.4 and 3.5 will not hold with non-negligible probability. Otherwise, the subset condition by Equation 3.6 or the completeness condition by Equation 3.2 will fail.

$C^*(Q) \neq C(Q)$ and $\Pi^* = \Pi$. In this case, the probability that Verify outputs *accept* is negligible because given the coefficients in Π^* equal to those in Π and different search results, the coefficient validity check will succeed only with negligible probability. Even if it passes this check, it will not satisfy the subset condition by Equation 3.6 or the completeness condition by Equation 3.2.

$C^*(Q) \neq C(Q)$ and $\Pi^* \neq \Pi$. If $d^*(r) \neq d(r)$, the Verify algorithm will output *reject*. If the integrity of the accumulation tree and the corresponding accumulation values is verified, and the coefficients in Π^* are computed correctly, $C^*(Q)$ will not satisfy the subset condition by Equation 3.6 or the completeness condition by Equation 3.2.

For the *private* Verify algorithm, we are also able to prove the *verifiability* property analogue to the above proof. □

In what follows, we prove the UC-security of our scheme.

Theorem 3.8. *The VCKS scheme is UC-secure if it satisfies* privacy *in Definition 3.2 and* verifiability *in Definition 3.3.*

Proof. (Sketch) If no parties are compromised by the adversary $\mathcal{A}$ in our protocol, for each keyword set Q, the user (P_1) outputs the correct search result $C(Q)$. Thus the environment $\mathcal{Z}$ cannot distinguish the real world from the ideal world since it only interacts with P_1.

If P_1 is corrupted by $\mathcal{A}$, then $\mathcal{A}$ can send the communication pattern of P_1 to $\mathcal{Z}$. In the ideal world, an adversary $\mathcal{S}$ can run $\mathcal{A}$ internally by playing the role of server (P_2). All messages between $\mathcal{Z}$ and $\mathcal{A}$ are forwarded by $\mathcal{S}$. $\mathcal{Z}$ cannot distinguish the real world from the ideal world since it will not interact with P_2 and $\mathcal{S}$ can play the role of P_2 faithfully.

If $\mathcal{A}$ corrupts P_2 and P_2 breaks *verifiability* in Definition 3.3 with negligible probability, the ideal world adversary $\mathcal{S}$ can run $\mathcal{A}$ internally by playing the role of P_1. All messages between $\mathcal{Z}$ and $\mathcal{A}$ are forwarded by $\mathcal{S}$. As such, $\mathcal{S}$ acts as same as the simulator in the Definition 3.2 and the ideal functionality $\mathcal{F}$ will reveal $\mathcal{L}_1$ and $\mathcal{L}_2$ to $\mathcal{S}$. From the proof of the Theorem 3.6, we can see that the inputs to $\mathcal{A}$ are distinguishable from those in the real world. In other words, $\mathcal{A}$ in the ideal world behaves as same as in the real world. On the other hand, $\mathcal{Z}$ cannot distinguish the outputs of the user in the real world and in the ideal world from the proof of the Theorem 3.7. For a search query, if $\mathcal{A}$ returns the valid ciphertext of the search result and its proof, the user will output correct plaintext of the search result in the real world, and in the deal world, $\mathcal{S}$ will return "accept" to $\mathcal{F}$ and $\mathcal{F}$ will send correct plaintext of the search result to P_1; otherwise, the user will output "reject", and $\mathcal{Z}$ will receive "reject" in the real world, and in the ideal world, $\mathcal{S}$ will return "reject" to $\mathcal{F}$, $\mathcal{F}$ will send "reject" to P_1 and $\mathcal{Z}$ will receive "reject" from P_1. For all the update queries, i.e., *modify*, *delete* and *insert*, the user receives nothing from $\mathcal{A}$. Therefore, she can always update the corresponding authentication information correctly and outputs nothing. Thus, $\mathcal{Z}$ cannot distinguish the real world from the ideal world. □

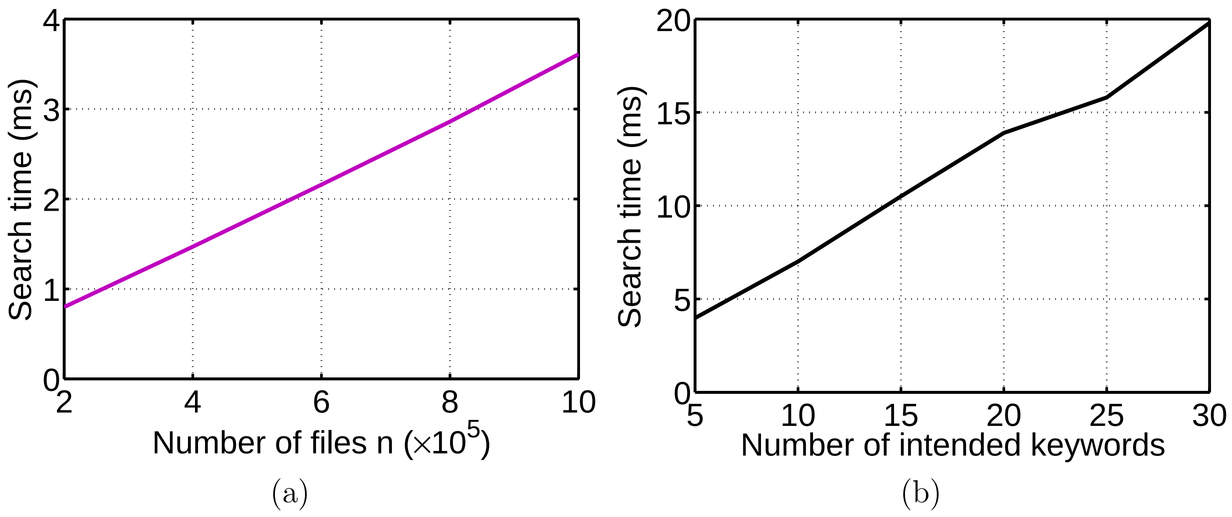


Figure 3.4: Search efficiency. (a) For the different size of file collection with the number of queried keywords $t = 2$; (b) For the different number of queried keywords with the number of files $n = 2 \times 10^5$.

3.7 Performance Evaluation

In this section, we evaluate the performance of our proposed VCKS scheme with real-world dataset, i.e., the Enron Email Dataset [32], which consists of about half million files. To demonstrate the efficiency and effectiveness of the VCKS, we extend the size of this dataset to one million by inserting duplicates but with different file identifiers. For simplicity, we set a two-level accumulation tree with $\epsilon = 0.5$. We conduct the experiment using C and the Pairing-Based Cryptography (PBC) Library [102] on a Linux server with Intel Core i7 Processor 2.4GHz. We adopt the type A elliptic curve of 160-bit group order to realize the symmetric version of our proposed scheme, which provides 1024-bit discrete log security equivalently. Our scheme can also be implemented by any other secure asymmetric pairing technique. The experimental result is an average of 10 trials.

3.7.1 Storage Overhead

Server side

The cloud server only stores the file ciphertexts C, the secure index γ and the accumulation tree T after the Setup operation by data owner. The storage overhead of C varies a lot for different file encryption method. Thus, we do not consider it here. For the size of γ, it is mainly determined by file collection size n and dictionary size m. As shown in Table 3.2, if there are one million files in the collection, the size of γ is linear to m. On the other hand, if m is fixed, the size of index is proportional to n as shown in Table 3.3. Our search verification scheme is storage-efficient, because Table 3.4 shows that the storage overhead of T with the fixed number of levels is only up to m, regardless of the dataset size n, and a minimal storage space suffices to host this tree structure.

Table 3.2: Size of encrypted index with $n = 1,000,000$.

m	1,000	2,000	3,000	4,000	5,000
Size of γ (MB)	125.03	250.06	375.10	500.13	625.16

Table 3.3: Size of encrypted index with $m = 2,000$.

n ($\times 10^5$)	2	4	6	8	10
Size of γ (MB)	50.06	100.06	150.06	200.06	250.06

Table 3.4: Size of accumulation tree with two levels.

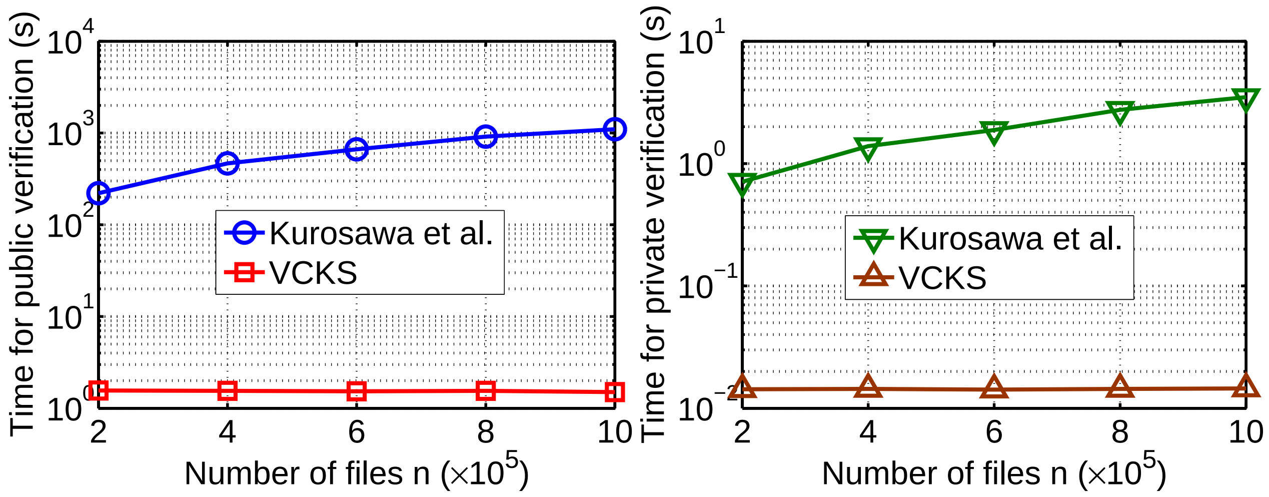
m	1,000	2,000	3,000	4,000	5,000
Size of T (KB)	66.1	103.9	195.6	260.1	324.6

Client side

The data owner and data users are all clients of the secure search system. In the *public* scenario, apart from the secret key s, data owner keeps the root node digest $d(r)$ after the accumulation tree generation phase and later sends it to users for search verification propose. She also retains the hash values $h(i, c_i)$ for all the files in C to efficiently update the file collection. In the *private* setting, data owner sends users s, $d(r)$ and $h(i, c_i)$ to enable efficient `Update` and *private* verification operations. The main storage overhead on the client side is to host all the hash values $h(i, c_i)$. We implement the hash function with SHA-256. Thus, for one million files, the size of their hash values are merely 32 MB.

3.7.2 Search Efficiency

The main computational cost for search process is to do the set intersection on t binary index vectors of size n with complexity $O(tn)$. We apply the bitwise `AND` operation to the queried keyword vectors $\{\delta_{j_b}\}$ for $b = 1, ..., t$. As shown in Figure 3.4(a), given the same two queried keywords, time cost for search is linear to file collection size n. In Figure 3.4(b), it shows that search is more time-consuming with the increased number of intended keywords. Experiment shows that our proposed VCKS scheme enables very fast conjunctive keyword search even with considerably large file collection. With a more powerful cloud server in practice, we expect that the search operation can be more efficient.



(a) (b)

Figure 3.5: Verification efficiency with $t = 2$. (a) Public verification; (b) Private verification.

3.7.3 Verification Efficiency

We evaluate the performance of the proposed verification mechanism in terms of the accumulation tree generation time, and result verification time in the *public* and *private* scenario.

Accumulation Tree Generation

Constructing the accumulation tree involves sum, multiplication and exponentiation operations in group G, and hash operation. Table 3.5 shows that with the dataset containing one million files, tree generation time is proportional to the dictionary size m. Notice that this computational burden on the data owner is a one-time cost. After the accumulation tree along with the encrypted index and dataset ciphertext is outsourced to the cloud server, the following operations, i.e., update and search verification, can be executed efficiently. Thus, the overall efficiency is totally acceptable in practice.

Table 3.5: Time of generating an accumulation tree with $n = 1 \times 10^6$.

m	1,000	2,000	3,000	4,000	5,000
Time (s)	618.95	1,237.89	1,856.84	2,475.78	3,094.73

Search Result Verification

Kurosawa *et al.* [67] designed a verification scheme for *single* keyword search by using RSA accumulator [12, 21], which also supports *public* and *private* verifiability. However, the verification complexity there is linear in the problem size $O(tn)$. To compared with our work, we adapt their scheme to be capable of conjunctive keyword search verification, i.e., after verifying each intended *single* keyword search, user will conduct conjunctive keyword search locally by index vector intersection. As shown in Figure 3.5(a) and Figure 3.5(b), our scheme can be orders of magnitude faster than theirs in both *public* and *private* settings. Moreover, the verification time is almost constant irrespective of n. In fact, the verification complexity of our scheme is $O(t + \rho)$, only decided by the related search operation, where ρ is number of files in the final search result. Therefore, our scheme is more suitable and practical for conjunctive keyword search over a large number of files stored in the cloud.

3.8 Summary

In a more challenging *malicious* model, we propose an efficient verifiable conjunctive keyword search scheme over large dynamic encrypted cloud data. Our scheme allows file update, i.e., users can insert, delete or modify a file without affecting the effective *conjunctive keyword search* operation. Furthermore, the verification cost is only contingent on the related search operation, regardless of the file collection size. Experimental result with a large real-world dataset shows the efficiency and practicality of our scheme. We also prove that the proposed scheme is *UC-secure* against a *malicious* adversary.

Chapter 4

Enabling Scalable and Efficient Range Query on Encrypted Genomic Data

4.1 Introduction

We are on the cusp of a new era in genomic research. Full genome sequencing (FGS) is conducted to comprehensively decode an organism's genetic make-up. It allows us to gain an unprecedented level of understanding on the biological inner workings. Medical/genomic researchers can now predict disease susceptibilities and drug responses base on a person's genome. Over the past decade, the cost for sequencing the genome of a person has been substantially reduced from \$100 million or so in 2001 to roughly $\$1,000$ in 2015 [80]. The promising future of personalized medicine is now within reach due to the sequencing cost reduction. On the other hand, one of the questions we need to answer is how one can effectively and efficiently handle the data proliferation as a result of FGS [121]. A large volume of genomic data from patients becomes a challenge for medical facilities. Public cloud, such as Google Genomics, DNAnexus, etc., may provide us with an economical solution for storage, but it also exposes users to the pitfall of privacy breach [98]. Due to the "highly-sensitive" nature of genomic data, such data loss can lead to severe consequences.

4.1.1 Secure Genome-wide Range Query

The human genome is composed of sequences of nucleotides. The process of human genome sequencing records segments of these sequences as short reads. Each short read contains genetic information of a biological property of an individual. Short reads are aligned to the human reference genome to determine its relative position in the chromosome. In medical research, scientists and medical professionals obtain short reads within a specific range in the chromosome to study the genetic attributes of an individual. This search is often referred to as *Genome-wide Range Query*. It retrieves all the short reads of interest *within the queried range* that is defined by a lower and upper position in *the entire genome*. Usually, such study is supervised or regulated by a Trusted Authority, such as National Institutes of Health (NIH), Food and Drug Administration (FDA). GRQ has been widely adopted in many applications in real world. For instance, a pharmaceutical company hopes to issue a query on a particular range of genomic data of a patient in order to find some DNA fingerprints/biomarkers for personalized medicine. In this case, the company needs to obtain approval on such query from the FDA first [9, 43]. For another example, an authorized physician can request a range of your genome for certain genetic tests that may indicate your potential response to a drug before he makes a clinical decision. With the low-cost DNA sequencing for the masses, Genome-wide Range Query given in the above examples is expected to be extensively used on top of the genomic data storage in many healthcare-related services and genomic research [7, 82].

Intuitively, existing secure range query techniques, e.g., order-preserving symmetric encryption (OPSE) [15], predicate encryption (PE) [74, 96], etc., may be the promising building blocks for a secure GRQ realization. However, directly applying them to an enormous amount of genomic data will cause privacy and efficiency concerns. In general, genomic data is range-sensitive because given specific query range information, the adversary can easily figure out the underlying genetic test. In order to facilitate efficient query, possible solutions [15, 74] in the literature may outright leak data ordering information which can be employed to compromise the range privacy. On the other hand, naïve use of current raw genomic data structure for multiprocessing will result in a significant scalability issue in the cloud [84, 118]. Direct adoption of the "heavy" cryptographic tools or an ill-designed index structure for the cloud deployment will even worsen the situation by introducing prohibitive performance penalty. In this work, our several key observations enable a novel solution to the efficient and scalable secure GRQ design. We first observe that the current secure range query solutions, when used in the context of GRQ, suffer from either slow search process [96], or compromised security guarantees [15, 17, 74], i.e. fully revealing ordering information, so as to achieve efficient query, let alone their scalability constraint. Straightforward as this order-comparison method is, it also gives the adversary a distinct advantage in the range identification with a reference genome.

4.2 Related Work

4.2.1 Cryptographic Range Query

Boldyreva *et al.* proposed an order-preserving symmetric encryption [15] that allows the ordering of the numerical plaintexts to be preserved in their encrypted forms. As a result, range query can be realized by ciphertext comparison. Apart from the practical security concerns [83], due to its order-revealing property, OPSE is not an appropriate building block for GRQ, which enables the adversary to determine the query range information trivially. Boneh *et al.* proposed an order-revealing symmetric functional encryption scheme [17], which is still inefficient for practical use due to the computationally expensive multilinear map operations. It also suffers from the order privacy breach as OPSE in the GRQ scenario.

Predicate encryption is another promising cryptographic primitive for GRQ. The decryption will succeed should the ciphertext fall into the queried range. In the public key setting, Shi *et al.* [97] proposed a PE scheme for multidimensional range query with linear search complexity. Note that the major inhibitor for the adoption of asymmetric PE in practice is the *predicate privacy* breach [96], i.e., an adversary can generate a ciphertext with the public key and then launch a brute-force attack to infer the encrypted query. Thus, people resort to the symmetric PE schemes [74, 96] to provide better search privacy at the price of significant usability deterioration in the case of GRQ (the strawman solution in Section 4.4.1). We may adopt a logarithmic-time B^+-tree based PE scheme proposed in [74]. However, for any ordered tree-based PE schemes, regardless of asymmetric or symmetric, the ordering information of the encrypted data will be directly disclosed. As a result, we can only achieve linear search when applying PE to GRQ at present.

4.2.2 Secure Keyword Search

Secure keyword search can be realized by either symmetric [34, 107, 109] or asymmetric [16, 110] cryptography. Curtmola *et al.* [34] proposed a searchable symmetric encryption (SSE) for single keyword search, and gave a formal security notion, i.e., CKA1 and a stronger and adaptive notion of CKA2. Sun *et al.* [109] proposed a UC-secure verifiable conjunctive keyword search scheme over dynamic encrypted cloud data in the malicious model. The verification cost only depends on search operation irrespective of the dataset size. In the public key scenario, Boneh *et al.* [16] presented a public key encryption with keyword search (PEKS) derived from identity-based encryption. In [110], the authors presented a verifiable attribute-based keyword search scheme, where only authorized users can obtain the intended search result.

4.2.3 Privacy-preserving Genomic Study

In the literature, privacy-preserving genomic data research has been extensively investigated to realize a variety of functionalities by the adoption of either secure multi-party computation (SMC) or secure outsourced computation techniques.

By using honey encryption, Huang *et al.* [58] proposed a secure outsourced genomic data storage scheme against the dictionary attack, where each decryption by the adversary even with an incorrect secret key will yield a valid-looking plaintext. Baldi *et al.* [9] presented a framework that aims for several important applications, such as paternity test, personalized medicine and genetic compatibility test by the generic secure two-party computation techniques. In [117], the authors presented an efficient secure edit distance approximation method, which is used to look for patients with similar genomic pattern (disease). Chen *et al.* [30] proposed a hybrid cloud-based scheme to delegate the partial read mapping task to the public cloud in a privacy-preserving manner.

There are few works towards secure and efficient solutions to the GRQ problem in the literature. The authors in [7] proposed a protocol that incorporates stream cipher, position scrambling and OPSE to store and retrieve the private raw genomic data. At the same time, the authors also expressed their concerns that this OPSE-based framework may not be secure for most practical applications [82], especially given a recent attack on OPSE [83]. Further, its practical efficiency is not satisfactory compared to ours.

4.3 Background

4.3.1 Biology Preliminaries

Genome

Genome represents the entirety of an organism's hereditary information, consisting of two long complementary polymer chains of four simple units called *nucleotides*, represented by letters A, C, G and T, which combine to form the double-stranded deoxyribonucleic acid (DNA) molecules in humans. There are approximately 3 billion nucleotides in 23 chromosomes of a human genome.

Single Nucleotide Polymorphisms

Single nucleotide polymorphisms (SNPs) are the most common form of DNA variation occurring when a single nucleotide (A, C, G, or T) in the genome differs between members of the same species or paired chromosomes of an individual. SNPs often correspond to how humans develop diseases and respond to pathogens, chemicals, drugs, vaccines and other agents. Thus, they are usually adopted as fingerprints in a variety of genetic tests and genomic research.

Raw Aligned Genomic Data

Raw aligned genomic data for an individual is the aligned output of a DNA sequencer, comprised of millions of *short reads*, covering the entire genome of that person and subsequently aligned by using a reference genome. Each short read corresponds to a sequence of nucleotides within a DNA molecule. The position of a short read with respect to the reference genome is determined by the approximate match on the reference genome [82].

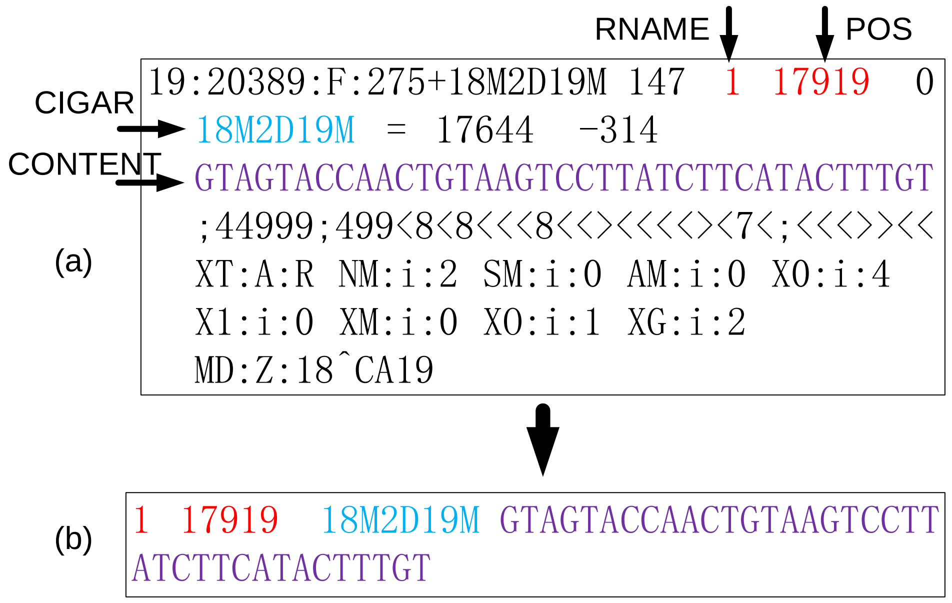

RNAME
POS
CIGAR
CONTENT
(a)
19:20389:F:275+18M2D19M 147 1 17919 0
18M2D19M = 17644 -314
GTAGTACCAACTGTAAGTCCTTATCTTCATACTTTGT
;44999;499<8<8<<<8<<><<<<><7<;<<<>><<
XT:A:R NM:i:2 SM:i:0 AM:i:0 X0:i:4
X1:i:0 XM:i:0 XO:i:1 XG:i:2
MD:Z:18^CA19
(b)
1 17919 18M2D19M GTAGTACCAACTGTAAGTCCTT
ATCTTCATACTTTGT

Figure 4.1: Format illustration of a sample short read. (a) The original format in a SAM file; (b) Simplified read format.

The raw alignment data can be stored in a sequence alignment/map (SAM) file, a BAM file (the binary version of SAM), or in a compressed CRAM file. Only SAM file is human readable albeit all the three formats contain the same alignment information [93] (We use SAM file for ease of illustration purpose only, but the proposed scheme is also applicable to BAM file). The format of a short read in a SAM file encompasses several data fields as shown in Figure 4.1(a), three of which are considered privacy-sensitive.

The first is the position information PI=(RNAME,POS). RNAME is the name of the chromosome where this short read resides, and POS is the position of where this read maps to the reference chromosome. For example, the POS of the read in Figure 4.2(a) relative to the reference is 5. As such, a *genome-wide range query* can be issued based on the positions PI of the intended short reads. For instance, with the range query $[(3, 100), (3, 150)]$, all the short reads, residing in between POS $= 100$ and POS $= 150$ of the reference chromosome 3, will be retrieved. The second sensitive field is the Concise Idiosyncratic Gapped Alignment Report (CIGAR) string, a sequence of nucleotide length and the associated operation, used to indicate which nucleotides align with the reference with letter `M`, are deleted from the reference with `D`, and are insertions not in the reference `I` (please refer to [93] for detailed exposition). Thus, the CIGAR string of the aligned read in Figure 4.2(a) is `3M1I3M1D5M`. The CONTENT field consists of sensitive nucleotide information, e.g., SNPs. On the other hand, the rest of a short read are deemed non-private fields for a person. Hence, we skip the discussion of these fields hereafter and only focus on the simplified privacy-sensitive read format as shown in Figure 4.1(b).

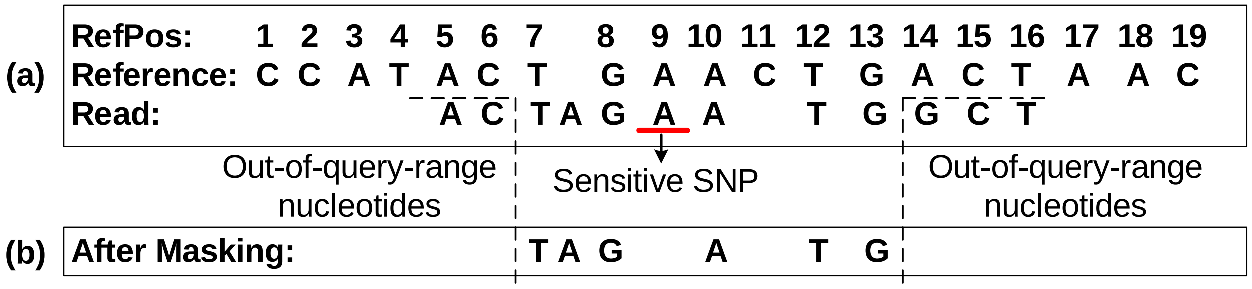

(a)
RefPos: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Reference: C C A T A C T G A A C T G A C T A A C
Read: A C T A G A A T G G C T
Out-of-query-range nucleotides
Sensitive SNP
Out-of-query-range nucleotides
(b)
After Masking: T A G A T G

Figure 4.2: (a) Short read alignment with the reference; (b) Content protection.

4.3.2 Secure GRQ Model

As shown in Figure 4.3, a typical real-world secure GRQ system consists of four entities, *data contributor* (DC), *medical unit* (MU), *trusted authority* (TA), and *Genobank*. Specifically, DC can be a patient who submits his biospecimen to a certified institution for full genome sequencing. Then his raw genomic data are generated and uploaded to TA for data preprocessing before outsourced to the public Genobank. Subsequently, TA anonymizes the data, e.g., it will erase the identity information linked to a particular patient. The anonymized data will be used to generate a secure index and then encrypted, which are sent by TA to Genobank in the end.

MUs including biomedical researchers, physicians, pharmaceuticals, etc., may ask for a particular range of the alignment data of a patient for further genetic tests (processing GRQ for multiple patients may follow the same procedure as in the single patient situation). In this case, a GRQ request is sent to TA, which contains the identities of MU and the queried patient, intended genome range and query purpose (what kind of genetic test to be conducted later). It is worth noting that *the role of TA has been established in the plaintext GRQ before our design* and can be played by government agencies, like NIH, FDA, etc., to perform the authentication/authorization to MU and its query. Upon authentication, a private GRQ request is produced, and TA submits it to Genobank on behalf of MU.

Genobank runs the secure GRQ search algorithm with the private GRQ query over the stored secure indexes. The corresponding encrypted alignment data is then retrieved and sent back to TA. In the end, TA decrypts the data, masks the sensitive information based on the genomic privacy policy, and returns result to MU. As with previous works [7, 30], we assume that TA as the only authoritative party in the system is anticipated to have sufficient computation resources (e.g., dedicated server, private cloud, etc.), or can delegate the computation to other certified institutions. Furthermore, all the communication channels in the system are assumed to be secure. The discussion on the corresponding genomic data de-identification [92] and user authorization in the search phase [34, 108] is outside the scope of this study.

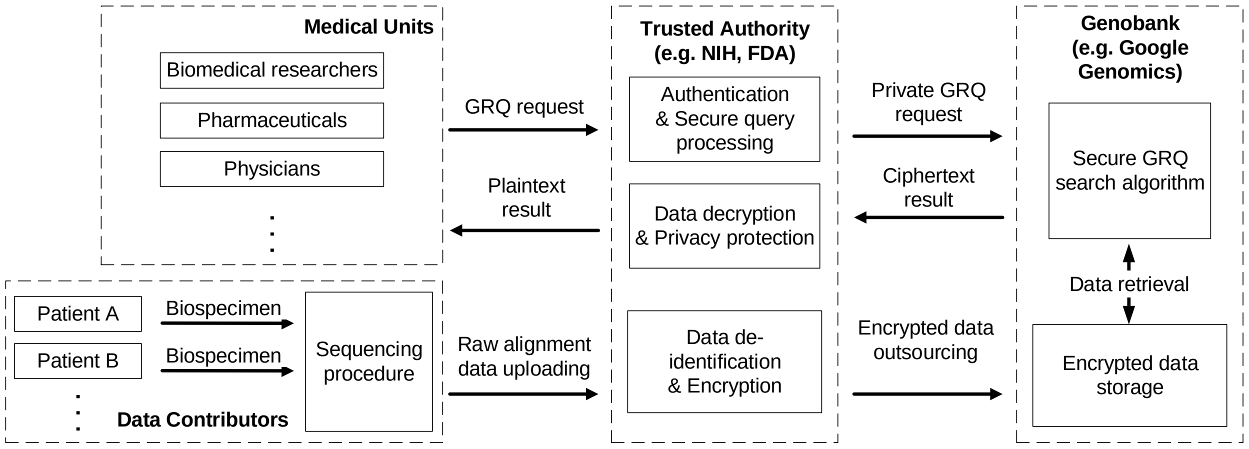

Medical Units
Biomedical researchers
Pharmaceuticals
Physicians
GRQ request
Plaintext result
Trusted Authority (e.g. NIH, FDA)
Authentication & Secure query processing
Data decryption & Privacy protection
Data de-identification & Encryption
Private GRQ request
Ciphertext result
Genobank (e.g. Google Genomics)
Secure GRQ search algorithm
Data retrieval
Encrypted data storage
Patient A
Patient B
Biospecimen
Biospecimen
Sequencing procedure
Data Contributors
Raw alignment data uploading
Encrypted data outsourcing

Figure 4.3: Overview of secure GRQ system.

4.3.3 Privacy Threats

We assume that DC is honest and not expected to be involved in the potential genetic tests [7]. TA is the key enabler entity in the system and acts as a private key manager/holder, standing between the user and Genobank. It conceals the identity of MU so that the semi-honest Genobank cannot infer the underlying genetic test through such side information.

We assume that MU will not collude with Genobank to gain unauthorized access privileges. For MU, as shown in Figure 4.2(b), we try to protect

1. *Out-of-query-range privacy*: We need to hide the *out-of-query-range* genomic information in the short reads of the result;

2. *Sensitive SNPs protection*: It requires the protection of sensitive SNP information *within* the queried range from MU, because these SNPs may happen to be indicators of other potential diseases irrelevant to the required genetic test For example, MU may need part of `F5` gene in chromosome 1 of a patient to learn his potential risk for stroke. However, the SNP `rs6025` within the requested range is also a well-known DNA fingerprint for the disease of venous thromboembolism [42] irrelevant to the required genetic test.

Almost all the secure range search constructions [7, 17, 74, 96] leak *access pattern*, i.e., after searching in the dataset, the encrypted result is disclosed, and *search pattern*, i.e., whether the range was queried before, if the underlying query generation is deterministic. We do not aim to protect them due to the incurred expensive computation or/and communication overheads[81].

We define two stateful leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ to precisely capture what is being revealed by the ciphertext and the query:

- $\mathcal{L}_1(\mathcal{IND}, \mathcal{F})$. On input of the index $\mathcal{IND}$ and the genomic data file $\mathcal{F}$ containing millions of short reads $\mathcal{SR}$, this function outputs the number of short reads in $\mathcal{F}$, and the size of each short read $|\mathcal{SR}|$;
- $\mathcal{L}_2(\mathcal{IND}, \mathcal{F}, \mathcal{Q})$. Besides $\mathcal{IND}$ and $\mathcal{F}$, this function also takes as input the query set $\mathcal{Q}$ searched in the past, and reveals *search* and *access pattern*.

We first adapt the CKA2 security in [34] for an MSSE scheme that serves the core construction for our GRQ protocol in Figure 4.5. Then we will reduce the security of the GRQ design to that of the MSSE (Section 4.5). Specifically, we aim to ensure the confidentiality of a short read and its position information relative to the reference genome during the query phase.

– Real-world Experiment $\textbf{Real}_{\mathcal{A}}(\lambda)$:
In the setup phase, the challenger runs $\mathsf{Setup}(1^\lambda)$ to generate a key set K.
$\mathcal{A}$ sends a tuple $(\mathcal{IND}, \mathcal{F})$ to the challenger. He receives $\bar{\mathcal{F}} \leftarrow \mathsf{Enc}(\mathcal{F}, K)$ and secure index structure $(\bar{\mathcal{IND}}_T, \{T_s\}) \leftarrow \mathsf{IndGen}(\mathcal{IND}, K)$.
In the search phase, for $i = 1, \ldots, q$,

- $\mathcal{A}$ chooses a keyword set $\mathcal{Q}_i$ and sends it to the challenger.
- The challenger returns an MSSE query $\bar{\mathcal{Q}}_i \leftarrow \mathsf{QueryGen}(\mathcal{Q}_i, K)$ to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a bit b.

– Ideal-world Experiment $\textbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$:
$\mathcal{A}$ chooses a tuple $(\mathcal{IND}, \mathcal{F})$. Given $\mathcal{L}_1(\mathcal{IND}, \mathcal{F})$, $\mathcal{S}$ outputs a tuple $(\bar{\mathcal{IND}}_T, \{T_s\}, \bar{\mathcal{F}})$ and returns it to $\mathcal{A}$.
In the search phase, for $i = 1, \ldots, q$,

- $\mathcal{A}$ chooses a keyword set $\mathcal{Q}_i$.
- Given $\mathcal{L}_2(\mathcal{IND}, \mathcal{F}, \mathcal{Q}_i)$, $\mathcal{S}$ returns $\bar{\mathcal{Q}}_i$ to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a bit b' in this experiment.

Figure 4.4: Real-world and ideal-world experiments.

Definition 4.1. (**CKA2 security for MSSE**) We consider two experiments in a real world and an ideal world respectively in Figure 4.4, where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator, and $\mathcal{L}_1$ and $\mathcal{L}_2$ are stateful leakage functions described above.

We say that the proposed MSSE scheme is CKA2 secure if for all PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$|Pr[\mathbf{Real}_{\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \text{neg}(\lambda).$$

4.4 Secure GRQ Construction

In this section, we first see how a conventional symmetric PE-based strawman solution fails the design goals (Section 4.4.1). Then we present our efficient and scalable secure GRQ scheme (Section 4.4.2) based on a novel hierarchical index structure and improve the search efficiency by filtering out the irrelevant query terms in a pre-search stage (Section 4.4.3).

4.4.1 Strawman Solution

PE has been adopted widely to achieve secure range query on outsourced encrypted data [74, 96], where only the ciphertext within the range of interest can be decrypted. Hence, seemingly we can design a secure GRQ scheme by trivially using symmetric PE for better *predicate privacy* [96]. However, when PE is applied to an enormous volume of data, such as human genome, efficiency and scalability become the primary concerns. For example, by using symmetric inner-product predicate-only encryption [74, 96], the encrypted index size for a single short read is proportional to the position domain size $N = 3 \times 10^9$, which leads to a significant ciphertext size expansion. Besides, the computation cost for encryption, token generation, and single read query operation are all proportional to N as well. Note that the dominant computation in the query phase is the composite-order pairing operation. Therefore, the service provider is not likely to favor such expensive computation approach. When $\Theta(N)$ pairing operations are required to search merely one short read, the scheme becomes unrealistic in a pay-as-you-go manner, even if the computation can be delegated to a more powerful cloud.

Last but not the least, the best search complexity we could achieve for this strawman solution is linear search so as to protect the query privacy, since using any sorted tree structure, e.g. B^+-tree [74], will outright leak the ordering of ciphertexts in the tree. Then the adversary can estimate the queried range relative to the reference genome, which may disclose the underlying genetic test.

Observations. We observe that the straightforward order-comparison methods used for the generic range query on encrypted data [15, 17] or logarithmic-time search over order-

Setup
$K \leftarrow \mathsf{Setup}(1^\lambda)$. On input of a security parameter λ, this probabilistic algorithm outputs a secret key set $K = (k_1, k_2)$.

Data Upload
$\bar{\mathcal{F}} \leftarrow \mathsf{Enc}(\mathcal{F}, K)$. To encrypt a short read $\mathcal{SR}_i$ in $\mathcal{F}$, TA generates a random nonce r_i, and then acquires the encrypted read $\bar{\mathcal{SR}}_i$ from the first l_i bits of $\mathcal{SR}_i^b \oplus h_{k_2}(r_i)$. $\oplus$ denotes the bit-wise XOR operation. In the end, all the short read ciphertexts $\mathcal{C}_i = \{\bar{\mathcal{SR}}_i, r_i\}$ for this patient are stored in a secure genomic data file $\bar{\mathcal{F}}$.
$(\bar{\mathcal{IND}}_T, \{T_s\}) \leftarrow \mathsf{IndGen}(\mathcal{IND}, K)$. Let $\mathcal{IND} = \{\mathsf{PI}_i\}_{1 \le i \le n_1}$ be an *ordered* index set for all the short reads in $\mathcal{F}$. Then TA first generates an obfuscated index set $\bar{\mathcal{IND}} = \{h(i)\}$ with $h(i) = h_{k_1}(\mathsf{PI}_i)$. Next, $\bar{\mathcal{IND}}$ is sorted to an ordered set as per the value of $h(i)$, which is further split into d partitions with equal length q such that $dq = n_1$. In addition, for each partition $s = 1, ..., d$, an ordered index table T_s is produced with q tuples in the form of $\{h((s-1)q + z), off_{(s-1)q+z}\}$, $1 \le z \le q$. $off_{(s-1)q+z}$ is the offset used to pinpoint the corresponding short read ciphertext. On the other hand, in each partition s, TA uses the highest ordering value $\{h(sq), pt_s\}$ to represent this partition range. pt_s points to the index table T_s. As such all the tuples for the d partitions constitutes a sorted range index $\bar{\mathcal{IND}}_T$ for d index tables. Finally, TA uploads $\bar{\mathcal{F}}$ along with the secure index structure $\bar{\mathcal{IND}}_T$ and $\{T_s\}_{1 \le s \le d}$ to Genobank.

Secure GRQ Search
$\bar{\mathcal{Q}} \leftarrow \mathsf{QueryGen}(\mathcal{Q}, K)$. On receiving a GRQ request with the range $[L, U]$ from an authorized MU, TA re-defines the query range $[L, U]$ to $[L', U']$ where $L' = L - a$, $U' = U + a$ and a is a random number greater than the size of the longest short read. Then he generates a query set, $\mathcal{Q} = \{\mathsf{PI}_i\}$ for $L' \le \mathsf{PI}_i \le U'$ and further the scrambled (encrypted) query set $\bar{\mathcal{Q}}$ using pseudorandom function h_{k_1} and a random permutation. TA submits $\bar{\mathcal{Q}}$ to Genobank on behalf of MU.
$L_{\bar{\mathcal{Q}}} \leftarrow \mathsf{Search}(\bar{\mathcal{Q}}, \bar{\mathcal{IND}}_T, \{T_s\}, \bar{\mathcal{F}})$. Upon receipt of $\bar{\mathcal{Q}}$, Genobank performs a binary search with each term in $\bar{\mathcal{Q}}$ on the sorted range index $\bar{\mathcal{IND}}_T$ which leads the query process to the relevant second-level index tables. Subsequently, Genobank conducts the binary search again on each related table T_s to retrieve the intended short read ciphertexts from $\bar{\mathcal{F}}$, puts them in the result list $L_{\bar{\mathcal{Q}}}$.

Data Download
$\mathcal{SR} \leftarrow \mathsf{Dec}(\mathcal{C}, K)$. TA decrypts each encrypted short read $\mathcal{C} = \{\bar{\mathcal{SR}}, r\}$ in $L_{\bar{\mathcal{Q}}}$ by $\bar{\mathcal{SR}} \oplus h_{k_2}(r)$ and checks whether PI of each plaintext $\mathcal{SR}$ falls in the range of interest $[L', U']$. TA also deletes the out-of-query-range nucleotides and the within-range sensitive SNPs in the CONTENT field as per the range $[L, U]$, and modifies the CIGAR string accordingly. Finally, TA returns the sanitized search result to MU.

Figure 4.5: Proposed secure GRQ construction.

revealed sorted data structure [74] will not apply to the range-sensitive GRQ service. To break such linkability, it requires us to look for a novel alternative approach, in contrast to the "off-the-shelf" methods, to protect the range information while realizing efficient query.

4.4.2 Our Construction

We propose a secure GRQ solution using a lightweight MSSE construction in Figure 4.5. It utilizes a scalable and efficient secure index structure, which offers not only genomic privacy preservation but also a logarithmic-time search complexity.

Suppose there are total n nucleotides in the reference genome and n_1 short reads in a patient's genomic data file $\mathcal{F}$. Thus, the universal position domain $\mathcal{P}$ contains n distinct PI. Let a short read $\mathcal{SR}_i = \{\mathsf{PI}_i, \mathsf{CIGAR}_i, \mathsf{CONTENT}_i\}$, $i = 1, ..., n_1$. Let $\mathcal{SR}_i^b$ be the l_i-bit long binary form of $\mathcal{SR}_i$.

Definition 4.2. (**Ordered Set**) We say $S = (s_1, s_2, ...s_N)$ of size N is an *ordered set* if its numerical terms are sorted by an ascending order such that $s_1 < s_2 < ... < s_N$.

Definition 4.3. (**Scrambled Set**) For an *ordered set* $S = (s_1, s_2, ...s_N)$, applying a pseudo-random function h_k with secret key k and a random permutation σ to S yields a *scrambled (encrypted) set* $\bar{S} = \{h_k(s_{\sigma(1)}), h_k(s_{\sigma(2)}), ..., h_k(s_{\sigma(N)})\}$.

Efficient Data Protection

Typically, storing the raw alignment data for a patient is significantly space-consuming, e.g., even with the compressed data format CRAM, the file size can be easily over 20 GB. The underlying data encryption is expected to be efficient in both storage and computation. As shown in Figure 4.5, the short read is XORed with a pseudorandom bit string produced from a random nonce. Subsequently, a collection of the encrypted short reads along with the corresponding random numbers and other encrypted auxiliary information in the original SAM/BAM file are all stored in one secure genomic data file $\bar{\mathcal{F}}$. This design is consistent with the real-world situation, where a single SAM/BAM file is used to store all the raw alignment data of a person. Another key observation behind this design is that storing each encrypted short read as an individual file can have an adverse impact on the system because hundreds of millions of small files present in the file system can degrade the cloud performance [118]. Moreover, the query process depends on the file storage lookup, which is often not specialized for range query problem.

It is worth noting that TA does not directly send the queried short reads back to MU after decryption. Instead, he re-writes part of the result to prevent MU learning additional information beyond the requested. To protect the *out-of-query-range privacy* and *sensitive SNPs* within the range (Section 4.3.3), TA modifies the $\mathsf{CONTENT}$ and the associated CIGAR

string of the corresponding short read. For example, in Figure 4.2(b), the CIGAR string is changed from `3M1I3M1D5M` to `1M1I1M1D1M1D2M` in accordance with the privacy policy.

```
Input: Preloaded sorted range index $\bar{\mathcal{IND}}_T$ for $d$ partitions with equal length $q$, in-disk
       table indexes $\{T_s\}_{1 \le s \le d}$, secure genomic data file $\bar{\mathcal{F}}$, and scrambled GRQ query set
       $\bar{\mathcal{Q}} = \{h_{k_1}(\mathsf{PI}_i)\}$ for $L' \le \mathsf{PI}_i \le U'$.
Output: GRQ query result $L_{\bar{\mathcal{Q}}}$.
1  Let $L_{\bar{\mathcal{Q}}} = \emptyset$;                                    ▷ Phase 1 query
2  for $j = 1 \to U' - L' + 1$ do
3      Perform a binary search with each element in $\bar{\mathcal{Q}}$ over $\bar{\mathcal{IND}}_T$ in memory;
4      Pinpoint the second-level index $T_s$ in disk with the corresponding pointer $pt_s$ in the
         partition range;
5      Load $T_s$ into memory;
6                                                                     ▷ Phase 2 query
7      Perform a binary search with $h_{k_1}(\mathsf{PI}_i)$ over $T_s$ in memory;
8      Find out the file offset in $T_s$ of the intended encrypted short read and use it to retrieve
         the short read from $\bar{\mathcal{F}}$ in disk;
9      Put the result in $L_{\bar{\mathcal{Q}}}$;
10 end
11 return $L_{\bar{\mathcal{Q}}}$;
```

Algorithm 4.1: Two-phase Query Algorithm.

Secure GRQ Request Processing

Upon receipt of the original GRQ request from DC, TA needs to transform the query to a secure version. Specifically, TA generates a random number a, greater than the number of nucleotides in the longest short read in the SAM file. He then derives a new query range $[L', U']$ from the original $[L, U]$ as in Figure 4.5 in the sense that a short read with its position in $[L', L-1]$ may also contain the nucleotides in $[L, U]$. To preserve the symmetry property, TA also modifies U to $U + a$. Then the query set size is $U' - L' + 1$. In the case of the query approaching the end of the genome, TA renders a an appropriate value that may be smaller than the maximum short read length. Finally, the queried range can be represented by an *ordered set* $\mathcal{Q}$ and later turned into a *scrambled set* $\bar{\mathcal{Q}}$ as the secure query set.

Scalable GRQ-oriented Index Structure in the Cloud

In order to enable large-scale privacy-preserving genomic research, it is imperative to store data in an efficient and scalable data structure. However, the current *de facto* genomic data organization is SAM/BAM file specification [93]. The binary BAM file is more data-processing friendly, but also has been criticized for its lack of scalability in a multiprocessing environment due to the use of a centralized header [84].

Our GRQ-oriented index structure involves a two-level design, focusing on efficiently retrieving the intended short reads from the secure genomic data file. As shown in Figure 4.6, a primary index $\bar{\mathcal{IND}}_T$ is devised and resides in the memory as the initial coarse-grained search phase. The benefit of this design is that it is not necessary to load the large index set $\bar{\mathcal{IND}}$ for a patient's entire raw alignment data into memory. Instead, a succinct index representation $\bar{\mathcal{IND}}_T$ is used to quickly pinpoint the relevant index range, which is organized as a secondary index table $\{T_s\}_{1 \leq s \leq d}$, kept in the disk storage. To avoid the small-file problem in the cloud [118], these second-level index tables can be stored in a file and fetched by the pointers in $\bar{\mathcal{IND}}_T$. T_s stores the exact mapping from each secure index to the file offset of the encrypted short read in $\bar{\mathcal{F}}$. Due to the saved space from $\bar{\mathcal{IND}}_T$, it is possible to load larger index tables into memory on demand after *Phase 1* search over $\bar{\mathcal{IND}}_T$ so as to enable fine-grained *Phase 2* search. As our proposed index structure are sorted by the secure index values, any efficient logarithmic-time search algorithm can be applied here. It is also worth noting that with the in-memory primary index, it is possible to enable large-scale concurrent lookups compared to the file-based indexing scheme currently used by BAM file. The pseudo code of the efficient two-phase query algorithm is shown in Algorithm 4.1.

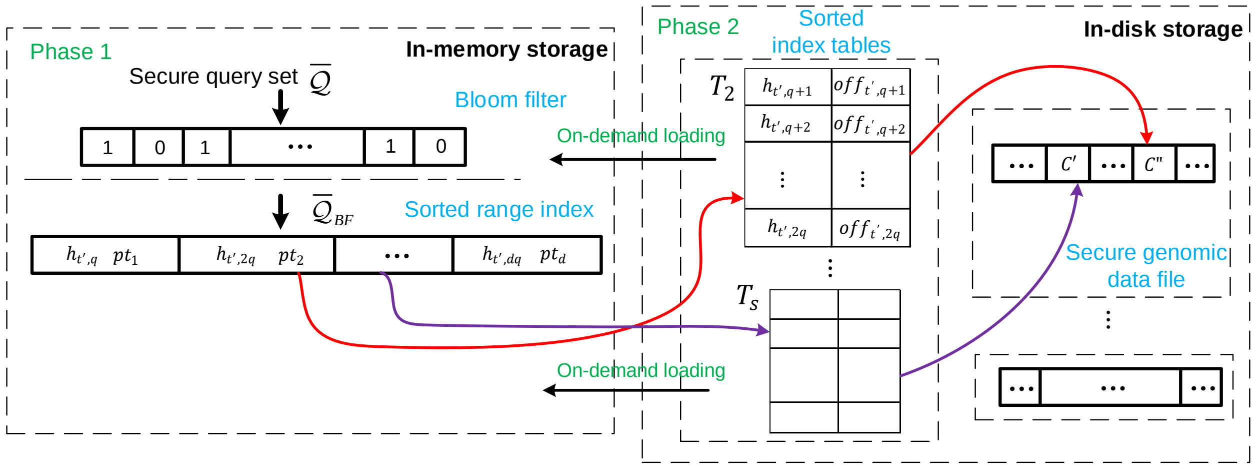


Figure 4.6: Hierarchical GRQ-orientated index structure with a two-phase query process.

Observations. The computation complexity of the proposed secure GRQ search is $\Theta(m \lg(n_1))$. The secure query set size m is possible to be as large as the size of the position domain $\mathcal{P}$, which means a considerable number of terms in $\bar{\mathcal{Q}}$ will not have a match in the following search process, thereby wasting the computation resources in practice. Should we pre-screen the query term before throwing it to the search process, a significant amount of time may be saved in a long run.

4.4.3 Improving Search Efficiency

Based on the above observations, we propose a pre-search stage to examine the existence of the query term to boost search efficiency as shown in Figure 4.6. More precisely, we adopt Bloom filter [13] on top of the existing secure index structure to provide efficient membership test. The Bloom filter is generated from the obfuscated index set $\mathcal{I}\bar{\mathcal{N}}\mathcal{D}$ with controlled false positive rate (FPR). By executing the membership test for terms in $\bar{\mathcal{Q}}$ prior to the two-phase query process, we can narrow $\bar{\mathcal{Q}}$ down to a likely smaller query set $\bar{\mathcal{Q}}_{\mathrm{BF}}$ taken as the input of the subsequent secure search algorithm.

Remark 4.4. Bloom filter will only introduce false positive but not false negative. This is desirable since no terms indeed belonging to the result will fail the membership test. Then we always have a complete search result. Furthermore, the false positive will not appear in the final GRQ result because such error can be rectified in the following exact-match search process. Notably, the membership test does not disclose the location of the short read in $\bar{\mathcal{F}}$. Linearly searching the entire file is prohibitively expensive. We still need to rely on our efficient search algorithm over the proposed genomic index structure to retrieve the short reads of interest.

4.5 Security Analysis

In this section, we show that our proposed secure GRQ scheme will achieve the defined security goals.

Theorem 4.5. *(Privacy against MU) The proposed secure GRQ scheme satisfies the out-of-query-range privacy and sensitive SNPs protection requirement for MU.*

Proof. (sketch) To prevent MU from acquiring the out-of-request information from the query result, TA re-writes the corresponding fields of the short read results before returning them to MU. More precisely, he modifies the CONTENT field to delete the irrelevant sensitive SNPs and truncate the short read as per the query range $[L, U]$. TA also produces a new CIGAR string accordingly. In the end, the *out-of-query-range privacy* and *sensitive SNPs* can be well protected against MU. □

The centerpiece of the proposed secure GRQ design is an MSSE construction in the sense that we may deem position information PI of each short read a unique keyword. Intuitively, the confidentiality of a short read and its position information are ensured if the underlying MSSE scheme is CKA2 secure. Therefore, the security offered by our scheme reduces to that of the MSSE protocol.

Lemma 4.6. *(CKA2 security for MSSE) The proposed MSSE scheme achieves CKA2 security [34], in the random oracle model defined in Definition 4.1.*

Proof. Let $\mathcal{A}$ and $\mathcal{S}$ be an adversary and a simulator in $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ in Definition 4.1, respectively. Assume that each $\mathcal{SR}$ in $\mathcal{F}$ has a unique PI. Given the leakage function $\mathcal{L}_1(\mathcal{IND}, \mathcal{F})$, $\mathcal{S}$ outputs $(\bar{\mathcal{F}}', \bar{\mathcal{IND}}'_T, \{T'_s\})$ as follows. It simulates each encrypted short read $\mathcal{C}'_i = \mathsf{Enc}(k_2, 0^{|\mathcal{SR}_i|})$ in $\bar{\mathcal{F}}'$ for $i = 1, ..., n_1$, where k_2 is randomly selected for the CPA-secure encryption algorithm Enc, and $|\mathcal{SR}_i|$ is revealed by $\mathcal{L}_1$. To simulate the secure index structure $\bar{\mathcal{IND}}'_T$ and $\{T'_s\}$, $\mathcal{S}$ first sets $h'(i)$ for each $\mathcal{SR}_i$ in $\mathcal{F}$ a random number, and then generates $\bar{\mathcal{IND}}'_T$ and $\{T'_s\}$ accordingly. The Bloom filter can also be simulated by using random numbers instead of cryptographic hashing.

Adversary $\mathcal{A}$ can make polynomial number of queries by picking a keyword set $\mathcal{Q}$ of t continuous position information PI for $t \geq 2$. The leakage function $\mathcal{L}_2(\mathcal{IND}, \mathcal{F}, \mathcal{Q})$ discloses $L_{\bar{\mathcal{Q}}}$ to $\mathcal{S}$. Given this, $\mathcal{S}$ can simulate $\bar{\mathcal{Q}}'$ by including $h'(i)$ for the short reads in $L_{\bar{\mathcal{Q}}}$ and assigning the remaining in $\bar{\mathcal{Q}}$ random numbers if $t > |L_{\bar{\mathcal{Q}}}|$.

Adversary $\mathcal{A}$ cannot distinguish $\bar{\mathcal{F}}'$ from $\bar{\mathcal{F}}$ in experiment $\mathbf{Real}_{\mathcal{A}}(\lambda)$ since Enc is CPA-secure. Due to the pseudorandom function h_k and cryptographic hash function, $\mathcal{A}$ cannot distinguish $(\bar{\mathcal{IND}}'_T, \{T'_s\})$ and $(\bar{\mathcal{IND}}_T, \{T_s\})$, and the Bloom filters. Likewise, $\mathcal{A}$ cannot discern $\bar{\mathcal{Q}}'$ and $\bar{\mathcal{Q}}$. Thus, $\mathcal{A}$ cannot distinguish $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ and $\mathbf{Real}_{\mathcal{A}}(\lambda)$. □

Theorem 4.7. *(Security for the GRQ scheme) The proposed secure GRQ scheme enjoys the same security guarantees as the underlying MSSE construction, i.e., Biobank is not able to learn any information about the content of the short reads and position information in the query and secure index during the search phase.*

Proof. (Sketch) This is inherited from the proof of Lemma 4.6, by which our secure GRQ scheme achieves the CKA2 security for MSSE. In other words, the adversary cannot deduce the content of an encrypted genomic file $\bar{\mathcal{F}}$, and the position information of the short reads from the secure index and query in the random oracle model. □

Impact of access pattern leakage. Indeed, to provide a practical GRQ solution, we need to accept some measure of leakage. Thus, one of our design goals is to achieve an acceptable balance between performance and leakage. Given that access pattern is disclosed after the query, we find that limited partial ordering information will be revealed to the adversary, which also applies to almost all the secure range query constructions, regardless of the underlying primitives, i.e. PE [74, 96] (the strawman scheme), OPSE [7], etc. Specifically, the ascending or descending data order will be inevitably exposed if a motivated adversary observes sufficient query results, but still he cannot determine which order is correct. For instance, consider 3-record dataset (a, b, c). Having observed two range query results (b, a) and (c, b), adversary can figure out the possible data orders, either (a, b, c) or (c, b, a). We should note that, albeit it is trivial to perform such attack on a relatively small dataset, it is unlikely that the query results would spread over the whole genome, instead of sporadic genetic hotspots in light of our very constrained knowledge on human genome at present. Compared to existing solutions, our secure GRQ scheme, as an initial attempt, enjoys logarithmic-time query process, and is considerably scalable and suitable for real-world cloud environment, with equivalent or better security guarantees. We may adopt "heavy" cryptographic tools on top of our protocol, such as Oblivious RAM [49], for mitigation but at the price of sacrificing the practical usability and efficiency [81].

4.6 Performance Evaluation

In this section, we implement the proposed secure GRQ scheme using real human raw alignment data in an approximately 32 GB CRAM file [1]. After decompression, the BAM file is about 44 GB, which contains more than 300 million short reads with the average read length of 100 nucleotides. We use JAVA and shell scripts on a Linux server with a 3.1 GHz AMD FX 8120 processor and 32 GB DDR3 memory. The server runs on a WD100ZFAEX hard disk with 1 TB storage and 64 MB disk cache. We use SHA256 to construct the Bloom filter in the pre-search stage. The experimental result is an average of 10 trials.

4.6.1 Storage Overhead

Ciphertext Expansion

We measure the storage overhead on Genobank for the encrypted raw alignment data of a patient and its secure index structure. Note that the bit-wise XOR encryption in our scheme introduces no ciphertext expansion even for the huge-sized genomic data. In practice, the encryption could be instantiated by CRT[AES]. Besides, an additional 1.26 GB storage burden comes from the 4-byte nonce for each short read in the dataset.

Table 4.1: Performance of the proposed secure GRQ scheme without pre-search stage.

d	q	Index (MB)		Indexing time (s)	Loading time (s)	Query time (s) with different range size						
		$\mathcal{I}\bar{\mathcal{N}}\mathcal{D}_T$	T_s			100	500	1,000	5,000	10,000	50,000	100,000
21,845	14,388	1	0.66	546	0.21	0.91	3.98	7.87	38.6	75.46	385.33	761.78
16,384	19,184	0.75	0.88	778	0.18	1.44	7.99	11.86	59.41	105.47	583.87	1,017.5
10,922	28,776	0.5	1.32	790	0.14	1.96	9.94	17.16	83.65	161.12	803.78	1,568.62
5,461	57,553	0.25	2.63	758	0.11	3.71	16.1	32.09	165.88	318.1	1,636.79	3,223.33
2,184	143,884	0.1	6.59	817	0.09	8.57	39.8	79.66	401.47	805.16	3,895.04	9,071.23
1,092	287,769	0.05	13.17	1,099	0.07	16.62	79.16	159.48	791.59	1,656.91	7,947.97	18,046.59

Scalability

The strawman construction will consume prohibitive space, i.e. 700 GB or so for each short read index since the ciphertext size is linear to the whole genome length, thereby crippling its usability in practice. For real-world implementation, it is straightforward to load the entire secure index $\bar{\mathcal{IND}}$ into cloud memory, which requires roughly 14 GB with our scheme. It is significantly smaller than the strawman solution but still not practical for processing multiple GRQ queries concurrently even with the cloud. On the contrary, using our hierarchical secure index design, the cloud memory consumption for the primary index $\bar{\mathcal{IND}}_T$ is orders of magnitude smaller than both the strawman and heuristic implementation. As shown in Table 4.1, suppose that storing $\bar{\mathcal{IND}}_T$ in memory for each patient costs 1 MB, and we can use up to all the 32 GB server memory, which allows hosting more than 300 thousand patients' primary indexes in the memory simultaneously and demonstrates the significant scalability of our scheme. The total size of the second-level table indexes $\{T_s\}_{1\leq s\leq d}$ is comparable to the secure index $\bar{\mathcal{IND}}$ in the heuristic implementation but stored in the disk and loaded into memory on demand. Moreover, as shown in Table 4.2, we can effectively reduce the size of the query by spending an additional storage space for the Bloom filter and eliminating the candidate query terms that fail the membership test, and therefore speed up the query operation.

Table 4.2: Size of Bloom filter over more than 300 million short reads in pre-search stage.

FPR	2.5%	6%	12%	25%	50%
Size (MB)	287.7	219.4	165.4	108.1	54.1

4.6.2 Time Efficiency

Encryption and Decryption

It takes less than 4 minutes for TA to encrypt more than 300 million short reads. This overhead is one-time and can be further amortized by parallelization. Decryption will be much faster since the query result usually is a much smaller subset of the whole genome.

Index Generation

This procedure also imposes a one-time cost to TA. The computation overhead varies with the patient's genomic data size and index construction parameter d. We use Linux shell script to perform several data preprocessing tasks, including extracting positions of short reads in the BAM file, generating their secure representation as well as sorting them in the index. It takes 36 minutes to preprocess the raw genomic data. As shown in Table 4.1, for larger partition number d, less time is required to generate the secure index. This variation is due to the use of object serializer in our Java code. The larger d is, the smaller each second-level in-disk index table is, thus requiring less memory maneuvering in JVM and resulting in the gain in processing speed. For the implementation of our scheme that does not make use of the object serializer function, we expect the performance penalty of a smaller d to diminish. On the other hand, generating Bloom filter in the pre-search stage needs nearly constant time, roughly 20 minutes, for different FPR, because the number of short reads to index for a person is constant.

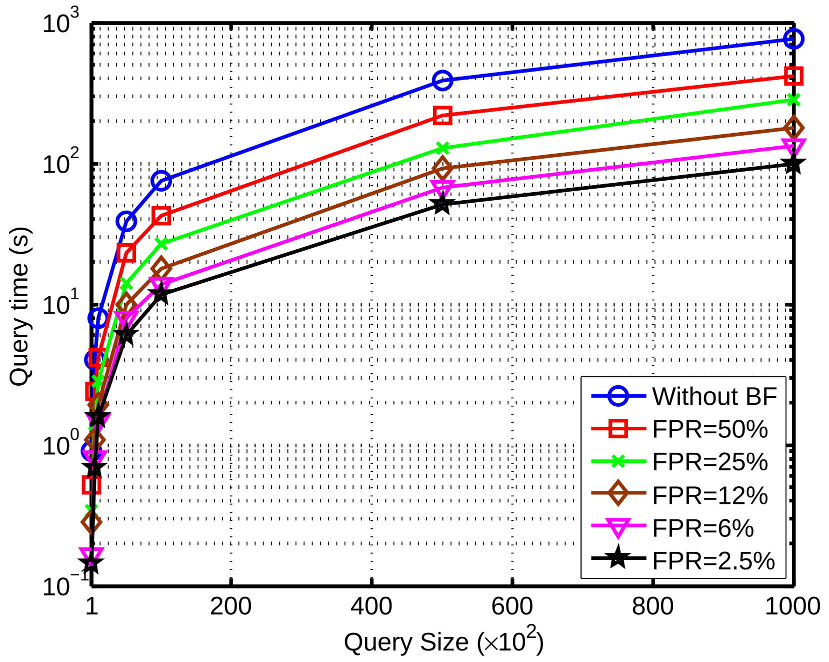


Figure 4.7: Query time for different search strategies with query size 100, 500, $1,000$, $5,000$, $10,000$, $50,000$, $100,000$. The pre-search stage is operated on the baseline search with $d = 21,845$.

Query Efficiency

Table 4.1 shows that query is more efficient with an increased d value. Larger d implies bigger first-level index, and thus a longer initial loading time. On the other hand, since the loading time for the primary index is less than 1 second, we believe that the load time penalty is not significant. After this one-time loading, $\mathcal{I}\bar{\mathcal{N}}\mathcal{D}_T$ resides in memory to facilitate expediting the query process. Moreover, a larger d triggers a more fine-grained first-level search. As a result, the on-demand secondary index loading is much faster due to its reduced size. We randomly select a GRQ query with different range size. The query time in Table 4.1 shows a linearity with the increased range size. We can see that our GRQ scheme is still efficient enough for practical use even with a large query range. In contrast, query time for the PE-based strawman solution is prohibitively expensive. Using the state-of-the-art benchmark for the costly composite-order pairing operation [123], it will take more than one year to search only one short read index. Notably, the OPSE-based GRQ scheme [7] reports a 4.5 second query time with a small 100 query range size. Lastly, we can also use Bloom filter in a pre-search stage to further ameliorate query efficiency. As shown in Figure 4.7, query time is significantly reduced by using the Bloom filter with a small FPR.

4.7 Summary

Genome-wide range query is one of the fundamental and critical components in genomic research, medical and healthcare services. In this work, we are among the first to propose a scalable, privacy-preserving GRQ scheme in the cloud environment, featuring an efficient hierarchical index structure. By presenting a novel MSSE-based secure range query scheme, our solution also enjoys storage and computation efficiency without sacrificing security guarantees. The implementation with real human raw alignment data shows its superiority over the existing solutions.

Chapter 5

Secure Keyword Search Using Trusted Hardware

5.1 Introduction

Nearly two decades has passed since Song *et al.*'s seminal work on the first encrypted data search scheme [100]. This demonstrated that the fascinating concept of retrieving information from encrypted data can be accomplished using cryptography. Since then, SE has received a growing interest from both academia [16, 18, 24, 34, 38, 62, 63, 66, 105] and industry [31, 99]. Recently, the importance of this technique has been highlighted due to the advent of cloud computing, where there is a strong desire to protect users' sensitive information from prying eyes while providing fundamental data services.

There are two main research directions in achieving the grand vision of search over encrypted data. One is software-based secure computation research, which often relies on cryptography and focuses on algorithmic design and theoretical proof. The other is the trusted execution solutions that depend on hardware isolation and trusted computing. The conventional SE is realized using software-based solutions. Albeit there are extensive investigations along this research line, current SE realization is not satisfactory in two aspects. First is the obvious query function gap between SE and the plaintext IR technology. This is because efficient practical SE solutions are built on top of a variety of crypto primitives, such as property-preserving encryption [15], functional encryption [16], and searchable symmetric encryption [34], and each crypto tool only supports a specific class of query types by incorporating different index structures and search algorithms. In addition, existing realistic SE solutions have many security limitations. In the symmetric setting, the most secure SSE we can achieve is under the L_1 leakage profile [25], which at least reveals the index search trace, including search pattern and access pattern (see Section 5.2). Unfortunately, the once considered "inconsequential" information disclosure has not been well studied and already led to many devastating attacks in practice [25, 59, 124]. Besides the above information leakage, public-key based schemes are inherently vulnerable to predicate privacy breach [96], i.e. an adversary can generate ciphertexts with the public key and infer the queried keywords during the search process.

On the other hand, hardware-based trusted execution environment has recently emerged as an effective security mechanism in achieving trustworthy execution of applications [3, 85, 94]. These systems adopt trusted hardware, such as Trusted Platform Module (TPM) [52], Intel Trusted Execution Technology (TXT) [51], ARM TrustZone [5], Intel Software Guard Extensions (SGX) [57, 77] and a small size of firmware as the trusted computing base (TCB). This TCB provides not only the root of trust but also the necessary system isolation for the environment. While it might appear that one can simply migrate the state-of-the-art IR techniques into the TEE to enable the same spectrum of query functions with enhanced security, there are several challenges that require careful design considerations to take advantage of the technology.

While the hardware-based secure execution, such as Intel SGX, can provide confidentiality and integrity of the application inside the TEE, information side channel is often not protected [19, 120]. The threat is greatly amplified when users share resources with adversaries, yet resource sharing is the basis of cloud computing. Recently, both control channel [120] and cache channel [19] have been demonstrated to leak execution information on the Intel SGX platform. Thus, direct adoption of TEE for secure search applications [8, 45] can lead to the disclosure of the index search trace. Another challenge lies in the programming environment. In order to defend against the untrusted operating system, each library function potentially needs to be redesigned to harden the defense against attacks, such as Iago attack [28]. At the time of writing, there are a limited number of library functions available in the enclave, the TEE of Intel SGX. None of the IR software we studied can be directly adopted as an enclave library due to missing libraries from the version 1.6 of the Intel SGX SDK.

In this work, we tackle the fundamental yet challenging problem of search over encrypted data. We propose REARGUARD, built on TEEs such as Intel SGX [57, 77] and AMD Memory Encryption [64] to perform search computation completely within the isolated memory even if the privileged software is untrusted. Such hardware-enforced isolation provides the confidentiality and integrity of both data and computation, which is essential in cloud computing. Furthermore, REARGUARD enables IR functions comparable to plaintext data search. Our scheme is also a departure from the pure software-based approach whose computation overhead largely depends on the underlying cryptographic primitives.

Current practical designs of software-based SE have the leakage profile that reveals at least the index search trace. REARGUARD can achieve better information protection and significantly improve the security of SE by mitigating such leakage. We define and realize two new leakage profiles, L_0^+ and L_0, in the dynamic SE scenario supporting index update. In the L_0^+ model, we completely hide the index search trace and only reveal minimal information at the setup as prior work. In the weaker notion L_0, we allow some reasonable leakage for a more efficient query. We identify several query-dependent operations in the search algorithms and adapt them to keyword-oblivious executions to satisfy the defined security requirements. We prototype REARGUARD with 4,000 lines of code (LOC). Considerable efforts are made to ensure that the implementation meets the security requirement while at the same time offering the desired query functions. REARGUARD provides search functionality and performance comparable to the plaintext data search.

5.2 Background

This section provides background information on current SE leakage, inverted index structure, and Intel SGX.

5.2.1 Privacy Leakage in SE

Among many cryptographic primitives for SE constructions, searchable symmetric encryption [34, 63, 105] is the most publicized and investigated in the literature for its "well" balanced privacy and efficiency tradeoff. SSE exploits deterministic encryption for efficient keyword match but its protection of keyword privacy is weak. Cash *et al.* defined four leakage profiles, i.e. L_4, L_3, L_2 and L_1, for SSE in the static setting to characterize the amount of information leakage [25]. Specifically, L_4 schemes reveal the number of words, their orders and occurrence counts in a document. Examples include some commercial products, such as Skyhigh Network [99] and CipherCloud [31]. L_3 profile reveals all the above information except occurrence counts of each keyword. L_2-SSE schemes only disclose the keyword number in a document. L_1 reveals the same information as L_2 but only for keywords that have been searched. Additionally, all the leakage profiles also imply the revelation of index search trace, including keyword *search pattern*, i.e. whether a query is repeated, and *access pattern*, i.e. pointers to encrypted files that satisfy the query. The majority of SSE adhere to L_2 or L_1 leakage. Such information disclosure also leads to *forward privacy* breach in the dynamic setting [18, 105]. This allows the adversary to learn whether the newly added document contains the keyword that has been queried before.

The consequence of the above leakage has not been well studied. Many attacks against SSE have emerged to exploit the leakage to partially or even fully recover the query and dataset information [25, 59, 124]. Unfortunately, these attacks are inevitable under the current security definition of SSE. Further, public-key based constructions are inherently vulnerable to predicate privacy leakage [96]. Namely, an adversary can generate ciphertexts with the public key and infer the query during the search process.

5.2.2 Inverted Index

Inverted index is widely adopted in modern search engines and current SSE design [34, 63, 66]. It enables sublinear search complexity (w.r.t. the number of files in the dataset), as well as

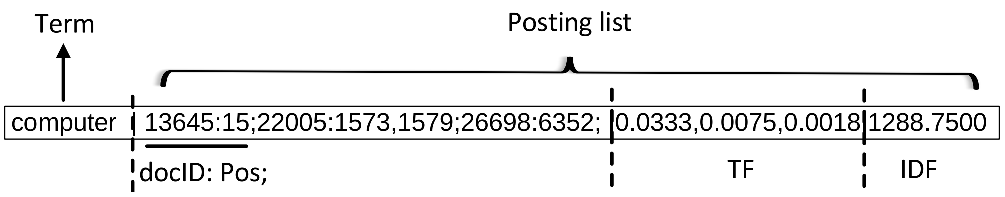


Figure 5.1: Example of an inverted index row.

rich query functions[1] in the dynamic setting, such as Boolean query, phrase query, ranked retrieval, spelling correction. It is generated by applying the standard indexing techniques to the target dataset [76], e.g. tokenization, stemming, stop words elimination, linguistic analysis, etc. In particular, this data structure contains N index rows vertically, where N equals the number of the extracted keywords (or terms, we use them interchangeably hereafter) from the dataset. Horizontally, it is divided into two major parts: a) the vocabulary (or dictionary) that includes all the extracted keywords, and b) the keyword-associated posting lists. Each list as shown in Figure 5.1 is composed of all the identifiers docID[2] of the documents containing the keyword, term position Pos in the corresponding documents, and other statistics, e.g. term frequency (TF) and inverse document frequency (IDF), etc. In general, a query is performed over the index by matching the target keywords (Step 1), retrieving the associated posting lists (Step 2) and evaluating query functions via information from the acquired lists (Step 3). Note that in Step 1, we can either linearly scan the vocabulary or use the hash table with constant search cost. The observed leakage (see Section 5.4) exists in both cases. For simplicity, here we choose to traverse the vocabulary for keyword match.

5.2.3 Intel SGX

SGX is the latest Intel's instruction extensions that aim to offer integrity and confidentiality guarantees to security-sensitive computation conducted on the commodity computer. The privileged software and low-level firmware, such as OS kernel, virtual machine hypervisor, and system management mode, are all assumed to be potentially malicious in the adversary model of SGX [57, 77]. This is because the TCB of SGX, compared to its predecessors, e.g., TPM [52], Intel TXT [51], only contains the CPU and several privileged *enclaves*. This significantly reduces the attack surface and provides strong security guarantees. The memory region reserved for the enclave, called enclave page cache (EPC), can only be accessed when CPU enters the special *enclave mode*, which enforces additional hardware checks on every external EPC page access request. The EPC memory is encrypted and authenticated by SGX Memory Encryption Engine (MEE) [33, 53], part of the memory controller within the CPU package. Enclave can save the encrypted computation result onto the untrusted persistent

[1] Most SSE works only support simplified versions of inverted index for extremely constrained query types, such as single keyword search.

[2] They can be pointers/URLs to the documents.

storage by using symmetric authenticated encryption, such as GCM[AES]. The encryption key is derived from the hard-coded *root sealing key* unknown to Intel. The ciphertext can be loaded back and decrypted later by the same enclave that encrypts it [4, 33]. In addition, SGX also provides the remote attestation function to convince users of the integrity of the established enclave before setting up the secure channel and provisioning their secrets [4, 60].

Besides physical attacks, SGX is also vulnerable to rollback attack [33], Iago attack [28] and side-channel attacks, including cache timing, power analysis, etc. SGX cannot defend against DoS attack as the underlying resource allocation is still controlled by the privileged system software. When applying these attacks to an SGX-based search, the adversary can interrupt the search process or/and infer the search privacy by breaking the SGX protection. This is out of scope of this study. Further, memory trace leakage has been confirmed at both page [120] and cache line level [19]. The leakage will disclose index search trace to the adversary in SE. We aim to mitigate such information disclosure in this work.

5.3 Problem Formulation

5.3.1 Overview

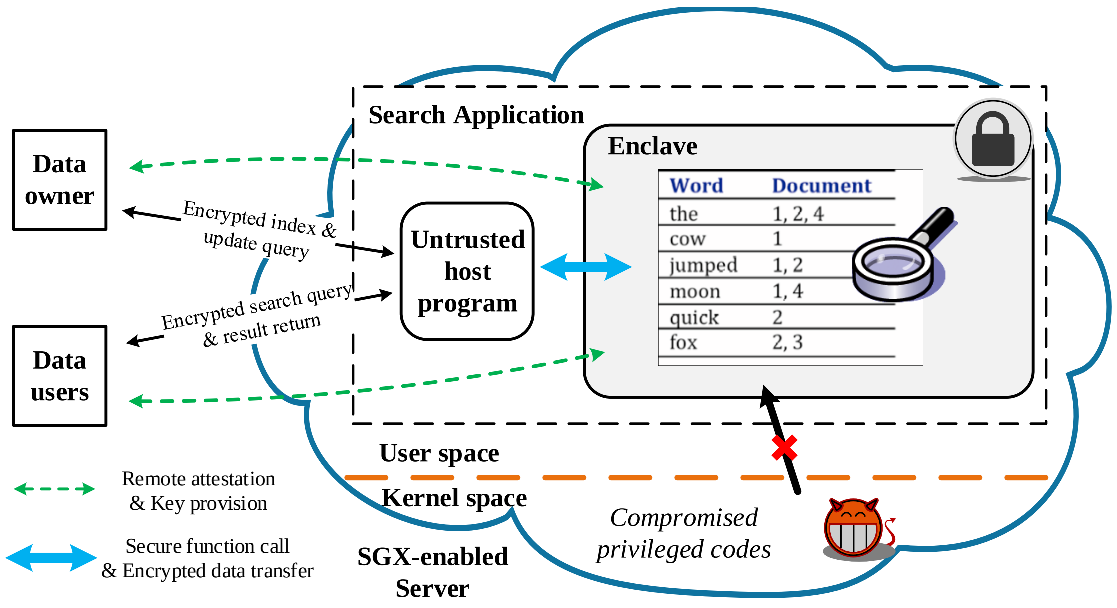
Three entities, *data owner*, *data user* and *SGX-enabled server* are involved in the system as shown in Figure 5.2. Data owner possesses a collection of n documents (we use "document" or "file" to refer to any text content records, such as text files, web pages), $\mathbf{DB} = \{D_1, D_2, ..., D_n\}$, which in turn contain m keywords $\mathbf{W} = \{w_1, w_2, ..., w_m\}$. Then he generates an inverted index $\mathcal{I}$ for $\mathbf{DB}$. Consistent with SSE [24, 34] we decouple the storage of the dataset from the storage of its index[3]. We assume that the code of query algorithms and associated parameters are public and have been preloaded into the enclave that is set up by the server. Next, the data owner authenticates himself to the server and launches remote attestation to check the integrity of the code and static data in the enclave. Then he establishes a secure channel and sends an owner-generated secret key sk_o to the enclave. The data owner also generates index ciphertext $\tilde{\mathcal{I}}$ under sk_o, for instance using GCM[AES], and pass it to the server. Later he, under the same sk_o[4], can issue an encrypted index update request τ_{upd} to add or delete a document.

By going through a similar procedure, an authorized data user i verifies the enclave that hosts the search code by remote attestation. This guarantees the integrity of the execution of the search program and correctness of the query result. The user also shares his secret key sk_{u_i} with the enclave and uploads the query ciphertext τ_s under sk_{u_i} to the server.

On the server side, search process begins with enclave loading and decrypting index $\tilde{\mathcal{I}}$ and

[3]This is a common practice. For example, Google is only responsible for index searching and maintaining, not hosting the actual web contents.

[4]A different secret key can be generated for each interaction with the server.


Search Application
Enclave
Data owner
Encrypted index & update query
Untrusted host program
Encrypted search query & result return
Data users
User space
Kernel space
Compromised privileged codes
SGX-enabled Server
Remote attestation & Key provision
Secure function call & Encrypted data transfer

Word	Document
the	1, 2, 4
cow	1
jumped	1, 2
moon	1, 4
quick	2
fox	2, 3

Figure 5.2: REARGUARD framework.

query τ_s with the corresponding keys. The query is executed over the plaintext index inside the enclave. The ciphertext $\tilde{res}$ of the result documents id using sk_{u_i} is sent back to user i. Another advantage of our design over SSE is that it naturally supports the more realistic multi-user setting because each user is able to search by his own secret key shared with the enclave.

Definition 5.1. (REARGUARD) Our secure keyword search scheme using trusted hardware is a tuple of three protocols executed between the data owner, data users and the SGX-enable server as follows:

- $(sk_o, \tilde{\mathcal{I}}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{I})$: On input a security parameter λ and an inverted index $\mathcal{I}$ for a dataset, it for the data owner outputs a secret key sk_o that will be shared with the verified enclave on the server and the encrypted index structure $\tilde{\mathcal{I}}$ that will be stored on the server.

- $(\tilde{res}, \tau_s, sk_{u_i}) \leftarrow \mathsf{Search}(1^\lambda, Q, \tilde{\mathcal{I}}, sk_o)$: On input the security parameter λ, it outputs for the user i a secret key sk_{u_i} that will be shared with the attested enclave on the server. Using sk_{u_i}, it encrypts a query Q on some keywords w into ciphertext τ_s, which is sent to the server. On input τ_s, $\tilde{\mathcal{I}}$, sk_o, sk_{u_i}, it outputs the search result ciphertext $\tilde{res}$ under sk_{u_i} for the user.

- $(\tilde{\mathcal{I}}_\Delta, \tau_{upd}) \leftarrow \mathsf{Update}(1^\lambda, upd, \tilde{\mathcal{I}}, sk_o)$: On input an index update request $upd = \{add/delete, w, id\}$ (perform addition or deletion of the file id over the posting list of keyword w) and the owner's secret key sk_o, it outputs an update token τ_{upd}. On input τ_{upd}, $\tilde{\mathcal{I}}$, sk_o, it produces an updated index $\tilde{\mathcal{I}}_\Delta$.

5.3.2 Adversary Model

We identify several keyword-dependent query operations and propose oblivious index access techniques to ensure that the adversary who observes a sequence of memory accesses toward the index, including the addresses and encrypted contents, has an indistinguishable view on index search (update) operations from the exhibited memory traces given two search (update) queries. Our security assumption is consistent with SGX except that we extend the side-channel attacks to comprise any attacks using information not derived directly from index access, such as target dataset statistics (e.g. posting list length, keyword frequency, etc.), context information (e.g. trending words, communication volume), and knowledge of linguistics. We do not intend to defend against them in this work. We also aim to achieve *query unlinkability*, i.e. the adversary cannot distinguish search tokens only by their appearances even for the same keywords, which is not supported by most SSE. In what follows, we first define two leakage profiles, L_0^+ and L_0 and then give our security definition.

Definition 5.2. (L_0^+ – Complete index access trace hiding) It reveals the initial index size at the setup phase, deterministic index search trace and update pattern of the same operation (add or delete) for any keyword.

Definition 5.3. (L_0 – Partial index access trace hiding) This profile reveals the initial index size at the setup phase, deterministic index search trace and update pattern of the same operation for keywords in the same group (see Section 5.4).

Similar to SSE, we do not consider index access operation types, i.e. search or update, to be sensitive information, albeit they can be further protected at extra cost [49].

Security Definition

We define the security of our scheme based on the simulation model of the secure computation [105]. In particular, it requires that a real-world protocol execution $\Pi_{\mathcal{F}}$ using the secure hardware functions be able to simulate an ideal-world functionality $\mathcal{F}$, such that an environment $\mathcal{Z}$, who produces all the input and reads all the output in the system, cannot distinguish these two worlds. We define the experiments $\mathbf{Real}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{F},\mathcal{A},\mathcal{S},\mathcal{Z}}(\lambda)$ in real and ideal worlds respectively as follows based on both leakage profiles L_0^+ and L_0.

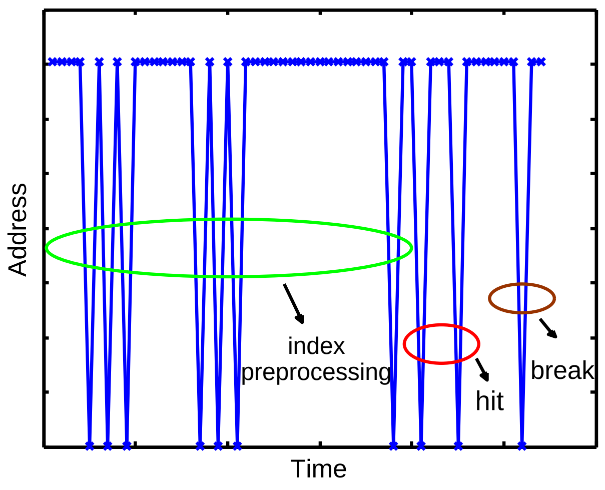
- $\mathbf{Real}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}}(\lambda)$: In the setup phase, an environment $\mathcal{Z}$ instructs the data owner by sending him a "setup" message to perform the Setup protocol with the real-world adversary $\mathcal{A}$. In each time step, $\mathcal{Z}$ specifies a search query Q for the user or an update request $upd = \{add/delete, w, id\}$ for the data owner. The user (owner) executes Search (Update) protocol. $\mathcal{Z}$ observes the protocol output for each search (update) operation, which is either a protocol abortion $\perp$, search result, or update success. Finally, it outputs a bit b.

- **Ideal**$_{\mathcal{F},\mathcal{A},\mathcal{S},\mathcal{Z}}(\lambda)$: In the setup phase, an environment $\mathcal{Z}$ sends the data owner a message "setup". Then the owner forwards this message to an ideal functionality $\mathcal{F}$, which notifies an ideal-world adversary $\mathcal{S}$ of the leakage L_0^+ (L_0). In each time step, $\mathcal{Z}$ specifies a search query Q for the user or an update request $upd = \{add/delete, w, id\}$ for data owner. The user (owner) submits Q (upd) to $\mathcal{F}$. Then $\mathcal{S}$ is notified of the leakage L_0^+ (L_0) associated with the search (update) operation by $\mathcal{F}$. $\mathcal{S}$ sends $\mathcal{F}$ either "continue" or "abort". $\mathcal{F}$ outputs either search result, update success, or $\perp$, which is observed by the environment $\mathcal{Z}$. Finally, $\mathcal{Z}$ outputs a bit b'.

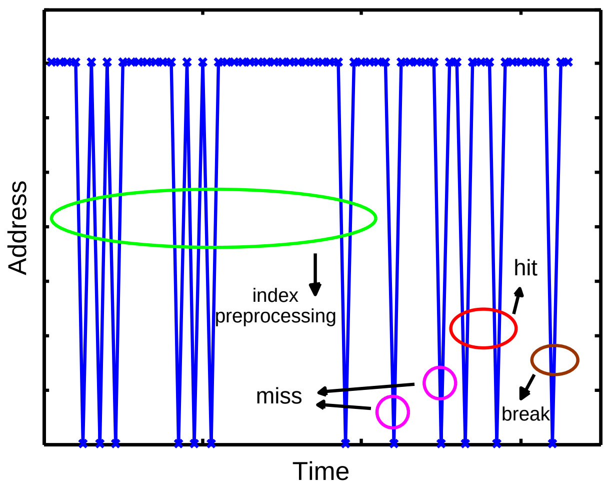
Definition 5.4. (Semi-honest/malicious security) We say that a protocol $\Pi_{\mathcal{F}}$ simulates the ideal functionality $\mathcal{F}$ in the semi-honest/malicious model, if for PPT semi-honest/malicious real-world adversary $\mathcal{A}$, there exists an ideal-world simulator $\mathcal{S}$, such that for all non-uniform, polynomial-time $\mathcal{Z}$,

$$|Pr[\mathbf{Real}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}}(\lambda) = 1] - Pr[\mathbf{Ideal}_{\mathcal{F},\mathcal{A},\mathcal{S},\mathcal{Z}}(\lambda) = 1]| \leq \text{neg}(\lambda).$$

Our security definition covers both the semi-honest adversary who faithfully follows the prescribed protocol and the malicious adversary that arbitrarily deviates from the protocol. Privacy of the scheme is guaranteed because $\mathcal{S}$ is only given the leakage L_0^+ or L_0 during the simulation. The definition also captures the correctness as data user or owner in the ideal world receives either the expected result or a protocol abortion.



(a)



(b)

Figure 5.3: Memory trace for keyword match in `Step 1`: (a) The first keyword match; (b) The third keyword match.

5.4 Our Design

This section provides concrete design for REARGUARD, especially focusing on the Search and Update phases. We first deal with the fundamental single keyword query, which is extensively studied in SSE. Then we describe the dynamic setting and consider the scalability issue with SGX. In the end, we discuss extensions to other common query functions, such as spelling correction, Boolean query, phrase query, proximity query, range query and similarity-based rank retrieval.

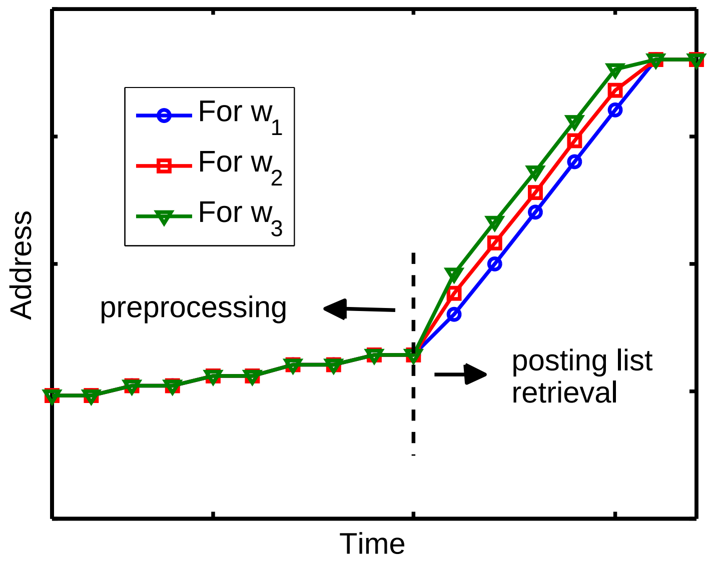


Figure 5.4: Memory trace for posting list retrieval in Step 2 for the first three keywords.

5.4.1 Single Keyword Query

Index Search Trace Leakage

The successful execution of a single keyword query over an inverted index returns file IDs within the posting list of the intended keyword. We identify two keyword-dependent operations in the single keyword search algorithm in Algorithm 5.1. The first sensitive operation is to search for the intended keyword w in the vocabulary of the index in Step 1 (see Section 5.2). If and only if the condition is met (line 3), the corresponding code block in Step 2 will be executed to further retrieve the posting list of this keyword (line 4) and then terminate the index search (line 5). We also experimentally verify the existence of the leakage for different keywords using Intel Pin framework [75] in Figure 5.3. Memory traces for different keyword matching in Step 1 are easily distinguished in Figure 5.3 (a) (b). If the first keyword is intended as in Figure 5.3 (a), the match hit can be observed followed by a break operation. In the case of the third keyword being the interest in Figure 5.3 (b), we will first observe

Input: Query keyword w and inverted index $\mathcal{I}$ including m keywords, where R_i is the ith index row. Define the data structure $Rindex$ for R_i, where $Rindex.term$ is the term for this row and $Rindex.plist$ is the corresponding posting list with data structure $Plist$.

Output: Query result res.

```
1 $res = \emptyset$
2 for $i = 1 \rightarrow m$ do
3     if $R_i.term == w$ then                // Keyword-dependent condition evaluation
4         $res = R_i.plist$                  // Keyword-dependent posting list retrieval
5         break
6     end
7 end
8 return $res$
```

Algorithm 5.1: Pseudocode for single keyword query.

two misses, then the match hit and the break in the end[5]. In Step 2 of posting list retrieval, we can also differentiate the index search patterns for different keywords effortlessly in Figure 5.4. The leakage may still exist at different granularities even using SGX[19, 120]. As a result, hiding memory traces of these two query-dependent operations plays a critical role in our design.

Oblivious Keyword Search Primitives

In a nutshell, we implement oblivious keyword search primitives to obfuscate the memory traces during index access. The main idea is similar to the techniques used in [85, 91] but we tailor them for the purpose of secure keyword search. Specifically, we realize the oblivious data transfer by X86 CMOVZ instruction, which moves the source operand to the destination operand if the condition code is true. When both source and destination operands are put in registers, this data transfer turns out to be oblivious and leaks no information about the branch selection. Likewise, we are able to use CPU registers as private storage to conceal the search footprints. For an *oblivious read*, we first load contents into registers and then merely select the data of interest. On the other hand, an *oblivious write* operation is carried out as follows. It first obliviously read the content. Should the data be intended, the updated content will be stored; otherwise, the original data will be written back. Since SGX uses randomized encryption to protect every write operation by MEE, the adversary cannot infer the data content. Moreover, we observe that the index search process consists of a sequence of read operations and that we only need to write the index at the update phase. Note that it is unnecessary to further obfuscate search and update operations similar to SSE, while

[5]The miss may not be observable if we use a hash table, but we can still capture the leakage by the address of the hit.

```
1  OMatch(Rindex* rind, Term qterm, Plist* tmp){
2  Plist* match;
3    __asm__ volatile (
4        "mov %3, %0;"
5        "cmp %1, %2;"
6        "cmovz %4, %0;"
7        : "=r" (match)
8        : "r" (rind → term), "r" (qterm), "r" (tmp), "r" (rind → plist)
9        : "cc"
10   );
11 return match;
12 }
```

Algorithm 5.2: OMatch() wrapper.

this can be done readily by additional dummy writes after oblivious reads.

Oblivious keyword match. We design an OMatch() function as shown in Algorithm 5.2 to hide the trace from keyword match by using the aforementioned oblivious data transfer primitive. In particular, we store both the query and keyword in the vocabulary in separate registers (line 8) and then compare them (line 5). If there is a match, the pointer to the posting list of the queried keyword will be returned; otherwise, it will return a default dummy address (line 6).

Oblivious posting list retrieval. Another oblivious function ORetrieval() is also adopted in order to hide the index search trace for retrieving the posting list in Step 2. A posting list will be obliviously retrieved only when its address matches that returned by OMatch(); otherwise, ORetrieval() function will return a dummy list. In addition, we can further improve the efficiency of array reading by the vector register AVX2 instead of element-wise read using a general purpose register.

Put All Together

We will show the concrete design for leakage profiles L_0^+ and L_0 respectively using the proposed oblivious search functions.

L_0^+ construction. The main idea behind the construction for L_0^+ leakage profile is to display the deterministic index search trace for each query. Specifically, we first use OMatch() to scan the entire vocabulary and obliviously match the queried keyword. Then ORetrieval() function is executed to obtain the posting list of the intended keyword after touching every index row. We further obliviously pad the retrieved list for every query to the predefined length l, $l \geq \text{MaxLength}(plists)$, so as to further obscure the attacker's view. Despite the complexity $O(N)$, our hardware-based scheme is efficient in practice because of the fast protocol execution between the CPU registers and DRAM of the server.

L_0 construction. Our intention of creating L_0 profile is to speed up the search process by tolerating extra information leakage compared to L_0^+ but still to achieve better security than SSE. We first randomly divide the keyword universe $\mathbf{W}$ into groups and scan the index until the group including the intended keyword has been searched, as shown in Figure 5.5. Next, the posting list of interest is obliviously retrieved from the group. This construction results in a faster query process than L_0^+. Efficiency can be further improved by using additional constant-overhead data structures, such as hash table, Bloom filter, to allow direct search over the target group. We also pad the result list from group i to a preset length l_{g_i}, which is not shorter than the longest list in the group. This L_0 design reveals which part of the index is being queried. However, the adversary cannot differentiate two queries for the same group through the disclosed aggregated group search pattern. L_0^+ can be considered a special case of L_0 with only one group – the entire index. We provide the detailed discussion on the implication of group size in Section 5.5.

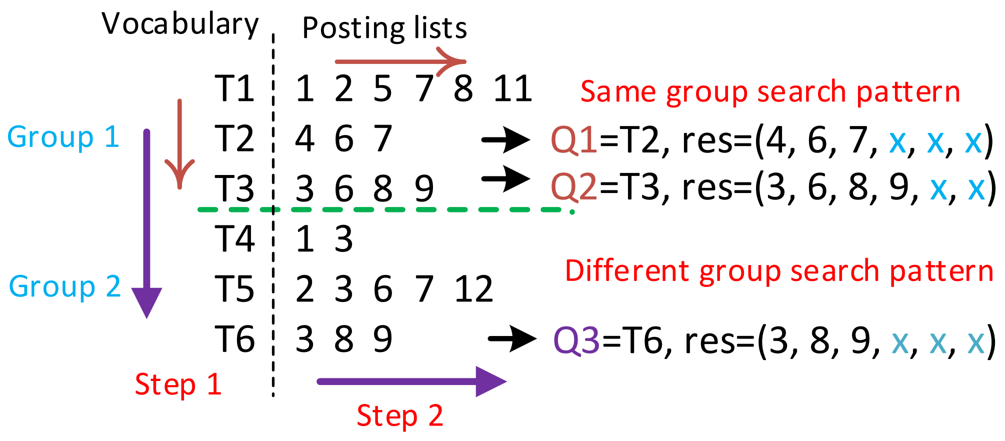

Vocabulary
Posting lists
T1 1 2 5 7 8 11
T2 4 6 7
T3 3 6 8 9
T4 1 3
T5 2 3 6 7 12
T6 3 8 9
Group 1
Group 2
Same group search pattern
Q1=T2, res=(4, 6, 7, x, x, x)
Q2=T3, res=(3, 6, 8, 9, x, x)
Different group search pattern
Q3=T6, res=(3, 8, 9, x, x, x)
Step 1
Step 2

Figure 5.5: Illustration for L_0.

Update

In the dynamic setting, the data owner is able to update the index by adding (deleting) a file to (from) the posting list of some keyword. We can leverage oblivious write operation to blur the view on the update. In particular, depending on the leakage profile, we first obliviously search over the index and obtain the intended posting list. For file addition, we insert the new file and its metadata to the retrieved list. Then we obliviously write the updated list back to the index, which increments the length of all the posting lists in the group for L_0 or in the entire index for L_0^+. Deleting a file for a keyword follows the similar procedure by replacing the target with a dummy file. In this case, the length of index rows is unchanged.

The type of update operation, *add* or *delete*, is also revealed to the server by observing the size of the updated index. Albeit we do not consider the leakage sensitive, we can foil it by intentionally writing a dummy file to the corresponding lists for the deletion operation.

The proposed approach may cause gradually increased index storage over time. To address this issue, the data owner downloads the index after a predefined number of update operations. He then refreshes the index by deleting the dummy files and randomly shuffles index rows. He also regroups the index for L_0 before encrypting and uploading it to the server. As a result, the adversary only observes an aggregated update pattern. The view can be further obfuscated by randomly cleaning a portion of dummy files in the index.

Scalability

The problem with the straightforward implementation of REARGUARD is that the EPC memory is constrained by current SGX specification, i.e. 128MB in total. Our experiment shows only about 95MB available for code and data. This scalability problem affects all applications built on Intel SGX at present. In the wake of indexing a large dataset, the index size is likely to exceed the limitation. We circumvent this pressing issue by splitting the original large index into small partitions at the setup. These partition indexes when sitting outside enclave are protected by authenticated encryption, and loaded into enclave on demand. For L_0^+, all partitions are sequentially loaded and searched inside the enclave. For L_0, we adopt a hierarchical index structure for efficient on-demand loading. Specifically, we put a small first-level index (e.g. using hash table, Bloom filter) into the enclave and use it to quickly pinpoint the second-level index partition in the main memory containing the target group. Then the enclave loads and obliviously searches over the partition. We are also able to achieve faster search by dividing the original index as per the groups. As such, only the intended group index is fetched by the in-enclave primary index.

5.4.2 Additional Query Function Support

Besides the single keyword query, plaintext IR compasses a variety of functions. Due to the page limit, we briefly describe how to incorporate some popular functions into REARGUARD design.

Spelling Correction

Spelling correction has a wide implementation in modern search engines to provide users with correct search results even in the presence of misspellings in the query. It proceeds by first computing the distance between the dictionary terms and the erroneous user input. Then it selects the keyword as query by some predefined value or let the user choose by providing him with spelling suggestions. Usually, the proximity is measured by edit distance (ED), k-gram overlap, or context information [76]. For example, given query `caa`, the suggested keywords within dictionary may include `cab`, `cat`, `car`, etc., with $ED = 1$. Note that we can incorporate this popular function into REARGUARD by modifying OMatch() function. Specifically, instead of exact keyword match, it checks whether the keyword being accessed is within the predefined distance to the user input. If it is true, this keyword is obliviously selected as the correct query. Alternatively, we can obviously constitute a candidate set as well that contains all the keywords within the range and let the user choose the best answer. In this case, it incurs additional overhead for the interaction.

Boolean Query

The Boolean query is another fundamental query type used in database and free text search. The query is formed by concatenating multiple keywords with logical operations, for example, (`computer` **OR** `network`) **AND** `security` **AND** (**NOT** `wireless`) and the result is expected to contain keywords {`computer`, `security`} or {`network`, `security`} but not `wireless`. Boolean query can be evaluated by performing set operations, i.e. intersection, union, and difference, in Step 3 based on the obliviously retrieved posting lists. In this phase, related data manipulation does not touch the index in this case. Thus, the index search trace will still be hidden. In addition, we may further enhance the security and curb the side information leakage by carrying out oblivious set operations that can be derived from the proposed oblivious data transfer primitives.

Range Query

A range query is extensively used in both database and free text search settings to match the records with queried terms within a certain range. We can either adopt a tree-based index, such as B-tree, k-d tree, in the enclave with equivalent security level of SSE [45], or transform the range query to a Boolean query [38, 111] so as to achieve L_0/L_0^+ security. With the latter, we do not need to alter our base index structure and seamlessly support this query type.

Other Query Functions in Step 3

We observe that many query functions are carried out in Step 3, the post index access phase. For example,

- **Proximity and phrase queries** are common query types. They can be treated as special cases of Boolean search with **AND** operations [76]. In phrase query, all the matched documents should contain a particular sequence of keywords while proximity query constrains the result by specifying the allowed distances between queried keywords. We leverage the physical position information of the queried keywords in the retrieved posting lists to measure the distance, e.g. the number of intermediate words or characters, in both query types. Thus, akin to Boolean query, we are able to make these Step-3 query types search trace leakage free.

- **Similarity-based rank retrieval** is an advanced IR technique to rank result files by their relevance to the query using statistics of the dataset, for instance, the "TF $\times$ IDF" weight in the cosine measure of the vector space model [106]. We can calculate the similarity score after the posting lists are obliviously retrieved from the index.

5.5 Security Analysis

In this section, we prove the security of REARGUARD for single keyword search. The proofs for other query functions are similar due to the same protection methods.

Theorem 5.5. *REARGUARD under L_0^+ is secure against the semi-honest adversary under Definition 5.4 if the underlying SGX primitives are trusted and encryption is CPA-secure.*

Proof. (Sketch). In the setup phase, $\mathcal{S}$ can output an index $\mathcal{I}'$ with randomly generated index rows as per L_0^+. Then it simulates the encrypted index $\tilde{\mathcal{I}}' = Enc_{sk_o}(\mathcal{I}')$, where sk_o is randomly selected for the CPA-secure encryption Enc.

In the search phase, according to L_0^+, $\mathcal{S}$ randomly selects a keyword w' from $\mathcal{I}'$ as query Q'. τ_s' can be simulated by $Enc_{sk_u}(Q')$, where sk_u is randomly produced.

In the update phase, simulator $\mathcal{S}$ outputs $upd' = \{op, w', id'\}$ based on the leakage function L_0^+. Specifically, op is either add or $delete$ as per the revealed update pattern. w' is randomly chosen from $\mathcal{I}'$. id' is also randomly selected accordingly. Then $\mathcal{S}$ sets $\tau_{upd}' = Enc_{sk_o}(upd')$.

As a result, the environment $\mathcal{Z}$ in Definition 5.4 cannot distinguish $\tilde{\mathcal{I}}'$, τ_s' and τ_{upd}' from $\tilde{\mathcal{I}}$, τ_s and τ_{upd} in the experiment $\mathbf{Real}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}}(\lambda)$ respectively due to the trusted execution environment enforced by SGX and CPA-secure Enc. $\square$

Theorem 5.6. *REARGUARD under L_0 is secure against the semi-honest adversary under Definition 5.4 if the underlying SGX primitives are trusted and encryption is CPA-secure.*

Proof. (Sketch). The proof for L_0 construction is similar to that in the L_0^+ model except that

- For search, $\mathcal{S}$ randomly selects a keyword w' in the revealed group from L_0.

- For update, w' is randomly chosen from the revealed group given the group access pattern leakage by L_0.

Thus $\mathcal{Z}$ cannot distinguish $\tilde{\mathcal{I}}'$, τ_s' and τ_{upd}' from $\tilde{\mathcal{I}}$, τ_s and τ_{upd} in the experiment $\mathbf{Real}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}}(\lambda)$ respectively, due to the secure hardware and CPA-secure encryption. □

In addition, we are also able to realize the security against the malicious adversary by proving the verifiability of the scheme. In general, this can be done through the remote attestation of SGX and replacing the CPA-secure encryption by authenticated encryption. Our design implies *forward privacy* (see Section 5.2) as well by hiding the memory trace of index update operation. Moreover, REARGUARD also achieves *query unlinkability* by using semantically secure encryption for the search and update token generation.

Privacy implication of group size in L_0. Attacks on SSE rely on precisely disclosed keyword search and access patterns during the index search phase [25, 124] to uniquely identify the queried keyword and speculate the plaintext dataset information. However, L_0 only leaks the aggregated group search/update pattern, including the number of keywords and length of their associated posting lists in the group. The adversary only knows if the queries are from the same group. The probability of precise keyword-query linkage is $1/n$ for an n-term group. Only given this, the adversary has no advantage in compromising query privacy except for random guessing as we have already proved. Therefore, regardless of the group size, the probability of revealing a query is exactly $1/|\mathbf{W}|$, as same as L_0^+. On the other hand, the adversary may exploit side information, e.g. the lengths of posting lists, context, communication volume, etc., to facilitate query identification. These side-channel attacks are hard to defend even with fully homomorphic encryption [46] and oblivious RAM (ORAM) [49]. The group size makes no differences in this situation.

5.6 Implementation and Evaluation

5.6.1 Implementation

In order for Intel SGX to offer its strong security guarantees, libraries included in the enclave has to be carefully designed to defend against potential malicious attacks [28]. At the time of writing, SGX SDK (v1.6) supports only C/C++. Many popular search software packages in other programming languages, such as the JAVA-based Lucene, cannot be directly adapted in the programming environment. Further, SGX uses a customized version of C/C++ standard library that only provides a limited subset of functions compared to the standard C library for security reasons. Therefore, even applications written in C/C++ such as Clucene cannot be directly migrated. We developed our own implementation of the search functions under the SGX development environment. To further alleviate index search trace leakage, privacy-sensitive operations are written using the memory-trace oblivious primitives. We built a prototype of REARGUARD with about 4,000 LOC using Intel SGX SDK v1.6 on Intel NUC, running Ubuntu 14.04 TLS. The NUC is powered by Intel i7-6770HQ Skylake CPU with 6MB cache at 2.6 GHz and 8GB DRAM. According to the vendor specification, the read and write speed of the 256GB SSD is 560 MB/s and 400 MB/s respectively. The AES encryption and decryption are implemented with Intel AES-NI instruction. This prototype supports common key query types and functions, i.e. spelling correction, Boolean query, proximity query, phrase query, range query (by converting to Boolean query) and similarity-based ranking.

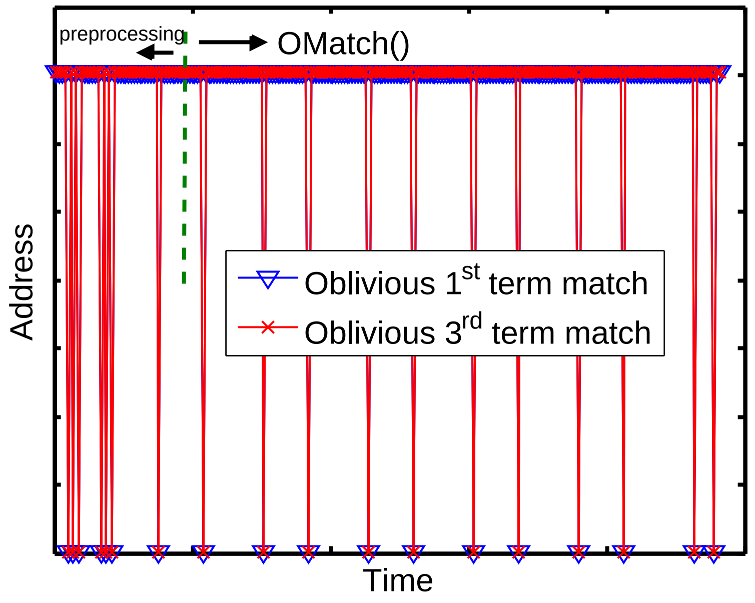


Figure 5.6: Oblivious keyword match in Step 1 during searching over a 5-term index.

5.6.2 Performance Evaluation

The objective of our evaluation is to measure the performance of the proposed system with elevated security protection. Existing SE work supports only a subset of the query functions we implemented. Plaintext and SGX-only searches are used as the baselines and compared to the proposed schemes in both L_0 and L_0^+ leakage models. For plaintext search, we evaluate its performance over the index entirely hosted in the main memory. SGX-only search is conducted using SGX protection but without oblivious operations, which can be generally deemed an L_1-SE scheme similar to [45]. Furthermore, we are interested in evaluating the scalability of the proposed system when handling a large-sized index that cannot be completely loaded into the EPC memory.

The experiments are conducted with a real-world dataset – Enron Email Dataset [32], which contains about half million files and has been extensively employed to evaluate SE schemes [24, 63]. We extracted about 258, 000 keywords and generated a 175MB inverted index after standard term stemming and stop word elimination. We use 80-keyword groups in L_0. The performance was measured over AND Boolean query, similarity-based ranking, and spelling correction (with default $ed = 1$) at the same time in all cases to demonstrate the efficiency and enriched query functions. We selected queries uniformly from the keyword universe. The experimental result is an average of 1,000 trials. For simplicity, we do not consider the optimization by advanced IR techniques, such as skip pointers for AND Boolean query and index compression, which are compatible with our scheme as we use the same index structure.

5.6.3 Oblivious Index Access

We first experimentally measure the effectiveness of the proposed oblivious keyword search functions. Figure 5.6 shows a unified view on index access in Step 1 by using OMatch() function regardless of the queried keywords in the vocabulary. This is applicable to both leakage profiles L_0^+ and L_0 with intended keywords in the same group. In L_0, search process by using ORetrieval() function in Step 2 exhibits, in Figure 5.7, the same index access pattern as long as the queried keywords are from the same group and distinct group search patterns otherwise. L_0^+ construction is expected to show an indistinguishable view on the posting list retrieval due to the deterministic index search pattern.

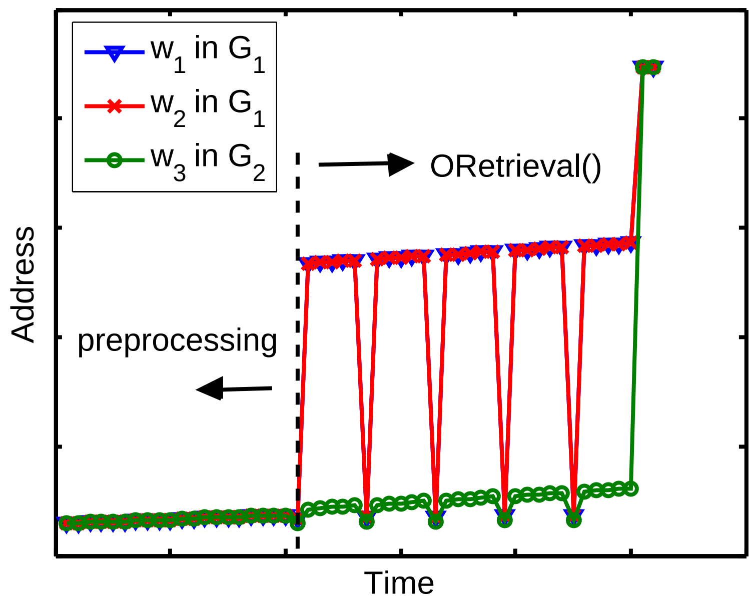


Figure 5.7: Oblivious posting list retrieval in Step 2 when searching over a 5-term index.

Search over Small-sized Index

We first measure the efficiency of searching over a small-sized index entirely residing inside the enclave. In our experiment, we only have less than 40MB EPC memory available. We randomly select a portion of the original index for all cases. The size of this index is about 35MB consisting of 50,000 keywords approximately. It is shown in Figure 5.8 (a) that time efficiency for all the cases is proportional to the number of keywords in the Boolean queries. We find that search inside enclave is very efficient and only costs about 0.62% additional time for encryption/decryption operations, context switching, etc., compared to plaintext search. On the other hand, REARGUARD are slightly slower than plaintext case due to the proposed oblivious index access functions. Although the L_0^+ design brings an $O(N)$ theoretical complexity, our experiment shows that only 1.16× overhead of plaintext search is incurred. The L_0 construction is faster than L_0^+ as expected and displays a 6% efficiency loss versus plaintext case. Note that 6 or fewer keywords in a query accounts for more than 96% cases in reality [65], therefore the experimental result in Figure 5.8 is a representative for the practical use.

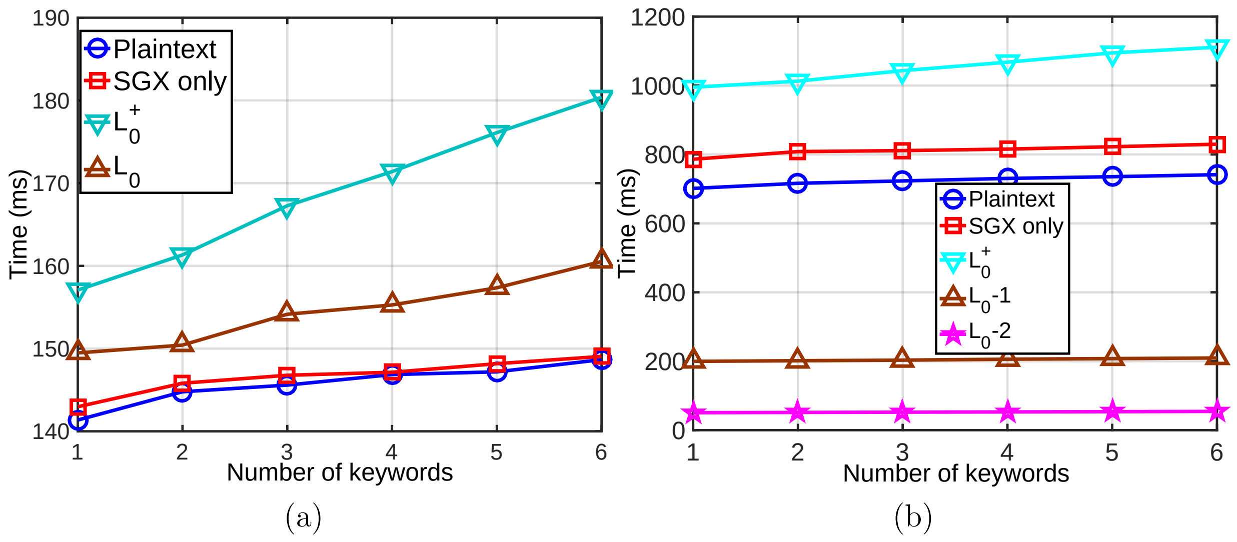


Figure 5.8: (a) Search over small-sized index. (b) Scalable search over large-sized index.

Search over Large-sized Index

For the index with size exceeding the available enclave memory, we divide it into partitions. We set up 5 partition indexes, each about 35MB, in our experiment. Plaintext search is still conducted over the original index in the main memory. The SGX-only search continues loading the index partitions until the match is found. Compared to the plaintext query, the SGX-only case in Figure 5.8 (b) shows an average 1.12× efficiency loss due to index loading, encryption/decryption, and context switching. We sequentially load and search over all the partitions for L_0^+, which is only about 1.45× slower than the plaintext case. Although plaintext search time can further speed up by optimizing the index structure, such as using our hierarchical index design, the absolute time cost of L_0^+ is still reasonable considering its strong security assurance. In addition to the overhead caused by SGX, the oblivious index access is the most time-consuming operation.

We split L_0 into two sub-cases. In L_0-1, we exploit a Bloom filter as the first-level index for each second-level index partition. We construct five Bloom filter indexes with all the false positive rate equal to 10^{-20}, which merely consume about 3.03MB EPC memory in total. When a hit is found in the Bloom filter index, we only load and search over the corresponding partition. In L_0-2, we build the Bloom filter for each group index with the same false positive rate as in L_0-1. The total size of the generated first-level index is about 3.3MB. Only the target group index is loaded and searched in the enclave in this case. In Figure 5.8 (b), both cases show nearly constant query time. L_0-1 is slightly slower than L_0-2 mainly owing to the relatively large index used there. Because we adopt a hierarchical index structure to allow search over smaller indexes, the two L_0 cases are faster than their competitors. The plaintext and SGX-only search are expected to be more efficient than L_0 with the similar index design.

Index Update

Updating index needs extra oblivious write operations compared to the search. Two update query types – *add* and *delete* – on the same index introduce almost the same cost. For an update query toward a group index in L_0, the experiment shows about 1.09× slowdown versus its counterpart search operation while L_0^+ introduces 1.1× time cost of the small-sized index search and 1.08× time cost of the large-sized index search.

5.7 Related Work

5.7.1 Search over Encrypted Data

Curtmola *et al.* [34] proposed the first searchable symmetric encryption scheme in the static setting. It gave two security definitions, i.e. CKA1 and CKA2, where the index search trace leakage is accepted for an efficient query. Kamara *et al.* [63] proposed a dynamic version of [34], supporting file insertion and deletion, but leaking forward privacy during the update. A forward-private SE was first explicitly considered in [105], where an ORAM-related technique was used to alleviate the privacy leakage but incurred non-negligible overhead. Later on, Bost [18] presented a more efficient forward-private SSE using trapdoor permutations. Boneh *et al.* [16] built the first public key encryption with keyword search from IBE. All the above SE works only support single keyword query. Recently, the secure Boolean query has been studied in the literature [24] but it still leaks index search trace. Another line of work focus on realizing secure range query. Many cryptographic primitives can be used as building blocks for efficient range query, such as order-preserving encryption [15], order-revealing encryption [17], predicate encryption [96], garbled RAMs [47]. Sun *et al.* [111] solved this problem by reducing a range query to a secure multi-keyword search in the genomic study scenario. Similar idea is also adopted in [38]. Recently a concurrent work by Fuhry *et al.* [45] was proposed to realize secure range query on an SGX-enabled server. However, the security is still consistent with current software-based SSE. Their scheme cannot be easily extended to support other IR functions either. In [8], a private database query scheme was proposed also using secure hardware. But it did not provide formal security analysis and their TCB is much larger than ours. Therefore, current SE only covers a small subset of plaintext query functions and cannot provide security guarantees beyond L_1 leakage while maintaining efficiency.

5.7.2 Applications with Secure Hardware

Recent years has seen increasing interest in building applications on top of secure hardware. Santos *et al.* [94] proposed a trusted language runtime using ARM TrustZone framework to protect the confidentiality and integrity of .NET mobile applications. In IoT setting, Ambrosin *et al.* [3] exploited secure hardware component to enable an asymmetric-key based swarm attestation protocol on IoT devices. There are recent efforts on harnessing Intel SGX to achieve security and privacy preservation for various applications. VC3 [95] was designed to realize the verifiable and confidential execution of MapReduce jobs in an untrusted cloud environment by using SGX. In [54], authors demonstrated the possibility of securely and efficiently evaluating functions in an SGX-backed trusted execution environment. Zhang *et al.* [122] proposed an authenticated data feed system based on SGX, which acted as a trustworthy proxy between HTTPS-enabled servers and smart contracts. Ohrimenko *et al.* [85] studied the problem of multi-party machine learning on an SGX-enabled server. Besides the confidentiality, they also considered the data-dependent memory trace leakage pertaining to the related machine learning algorithms.

5.8 Summary

In this work, we propose REARGUARD, the first secure keyword search scheme based on the off-the-shelf trusted hardware to achieve query functions comparable to plaintext IR while ensuring the confidentiality and integrity of the query process. We define two new privacy leakage profiles for SE and present corresponding constructions, which reveal much fewer search footprints than the state-of-the-art software-based solutions. We present approaches beyond the capability of the underlying hardware primitive by designing effective oblivious keyword search functions. Our implementation with the real-world dataset shows its practicality and efficiency.

Chapter 6

Conclusions

With the advent of cloud computing that is based on service-oriented architecture and virtualization technology, an unprecedented elastic computation model has been unfolded to us. We have seen more and more mission-critical applications along with a huge amount of potentially sensitive data from business, research and our day-to-day life migrating to the centralized servers, which inevitably leads to a powerful yet not fully-trustworthy public cloud in the pivotal position of the entire IT infrastructure. As this cloud brain governs various aspects of our society, a natural concern about its security and privacy has arisen. We target our research in this dissertation at this challenging public cloud environment, study and attempt to secure two fundamental cloud services – data deduplication in cloud storage and information retrieval in different applications and contexts. We, in this dissertation research, explore both software-based cryptographic approaches and hardware-based trusted computing technology, and strive to comprehensively exhibit the current landscape of the related problem solving in hopes of providing insights to the future research in this area.

Future Research Directions Besides the studied problems in this dissertation, there are other emerging applications and cloud-based architecture designs, which may need different considerations and demands for security and privacy. In what follows, we briefly introduce some interesting and important issues.

- Cloud data confidentiality and integrity can be achieved by using software/hardware-based solutions similar to those in this dissertation. However, different applications may require additional security/privacy considerations. For example, unlike the password or PIN-based authentication, user biometric data cannot be easily revoked or replaced, because they are permanently associated with the user. Once the immutable user bio-features are stolen from the server, all the applications using them as user identity in the biometrics-based authentication immediately expose to the compromise risk. Therefore, we must come up with new approaches to stay resilient to the breach and be able to safely *revoke* the bio-features of the user and *reuse* them for different applications later. Moreover,*unlinkability* is desired in the sense that the applications with the registered same set of user biometrics cannot be linked and boil down to the same user identity.

- Similar to the motivation of REARGUARD, we lack a unified generic solution to support a wide spectrum of cloud services. The respective enabling techniques in the literature inevitably cause inconsistency in the system and incur a noteworthy performance hit. TEEs may be an appealing option, but the scalability, heterogeneity, and usability make their application to secure generic computation tasks in different platforms an extremely challenging problem. There are continuing efforts to bridge the discrepancy, such as Google Asylo [88], but much more research is needed in this direction to explore other possibilities.

- We through the research of REARGUARDshow that the disclosure of the memory trace of the computation during the secure data service may reveal significant user privacy. This kind of side-channel threats are real (e.g. Meltdown and Spectre vulnerabilities in Intel CPU) and could be catastrophic. Therefore, early awareness and a much-deeper integration of oblivious system operations are desired instead of a separate remedial design that is likely to adversely impact the service performance.

- Another crucial function of the cloud brain, which we do not touch in this dissertation, is the reaction to the input data stimuli. This scenario is well modeled and demonstrated by cloud-based IoT systems. Data are constantly gleaned by IoT devices from the ambient environment and sent to the backend cloud for data processing and decision making. Accordingly, the cloud issues the control commands to the end IoT actuator devices to physically change the state of the environment or users, such as open the door, operate the medical device (e.g. insulin pumps). This setting obviously necessitates additional requirements in addition to the confidentiality and integrity at the data storage and processing phases. Specifically, we should have an accountable, auditable and verifiable cloud that is compliant with the service agreement and friendly to misbehavior detection, auditing, and forensics.

Bibliography

[1] 1000 Genome Project. ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/data/HG00097/alignment/, accessed in October 2015.

[2] Martín Abadi, Dan Boneh, Ilya Mironov, Ananth Raghunathan, and Gil Segev. Message-locked encryption for lock-dependent messages. In *Advances in Cryptology–CRYPTO 2013*, pages 374–391. Springer, 2013.

[3] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. SANA: Secure and scalable aggregate network attestation. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, pages 731–742. ACM, 2016.

[4] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, volume 13, 2013.

[5] ARM Security Technology. Building a secure system using TrustZone technology (white paper). *ARM Limited*, 2009.

[6] Man Ho Au, Patrick P Tsang, Willy Susilo, and Yi Mu. Dynamic universal accumulators for ddh groups and their application to attribute-based anonymous credential systems. In *Cryptographers' Track at the RSA Conference*, pages 295–308. Springer, 2009.

[7] Erman Ayday, Jean Louis Raisaro, Urs Hengartner, Adam Molyneaux, and Jean-Pierre Hubaux. Privacy-preserving processing of raw genomic data. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 133–147. Springer, 2014.

[8] Sumeet Bajaj and Radu Sion. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):752–765, 2014.

[9] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes.

In *Proceedings of the 18th ACM SIGSAC Conference on Computer and Communications Security*, pages 691–702. ACM, 2011.

[10] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In *Proceedings of the 22nd USENIX Security Symposium*, pages 179–194. USENIX Association, 2013.

[11] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology–EUROCRYP 2013*, pages 296–312. Springer, 2013.

[12] Josh Benaloh and Michael De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 274–285. Springer, 1993.

[13] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[14] Deepak R Bobbarjung, Suresh Jagannathan, and Cezary Dubnicki. Improving duplicate elimination in storage systems. *ACM Transactions on Storage*, 2(4):424–448, 2006.

[15] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam Oneill. Order-preserving symmetric encryption. In *Advances in Cryptology–EUROCRYPT 2009*, pages 224–241. Springer, 2009.

[16] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology–EUROCRYPT 2004*, pages 506–522. Springer, 2004.

[17] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Advances in Cryptology–EUROCRYPT 2015*, pages 563–594. Springer, 2015.

[18] Raphael Bost. $\Sigma o \varphi o \varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154, 2016.

[19] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. *arXiv preprint arXiv:1702.07521*, 2017.

[20] Andrei Z Broder. Some applications of rabins fingerprinting method. In *Sequences II*, pages 143–152. Springer, 1993.

[21] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology–CRYPTO 2002*, volume 2442, pages 61–76. Springer, 2002.

[22] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[23] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):222–233, 2014.

[24] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[25] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679. ACM, 2015.

[26] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of the 3rd International Conference on Applied Cryptography and Network Security*, volume 5, pages 442–455. Springer, 2005.

[27] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology–CRYPTO*, pages 199–203. Springer, 1983.

[28] Stephen Checkoway and Hovav Shacham. Iago attacks: Why the system call API is a bad untrusted rpc interface. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 253–264. ACM, 2013.

[29] Rongmao Chen, Yi Mu, Guomin Yang, and Fuchun Guo. BL-MLE: block-level message-locked encryption for secure large file deduplication. *IEEE Transactions on Information Forensics and Security*, 10(12):2643–2652, 2015.

[30] Yangyi Chen, Bo Peng, XiaoFeng Wang, and Haixu Tang. Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds. In *Proceedings of Network and Distributed System Security*. Internet Society, 2012.

[31] CipherCloud. Cloud data encryption. http://www.ciphercloud.com/technologies/encryption/, 2017.

[32] W. William Cohen. Enron email dataset. https://www.cs.cmu.edu/~enron/, 2017.

[33] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:086, 2016.

[34] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM SIGSAC Conference on Computer and Communications Security*, pages 79–88. ACM, 2006.

[35] Damballa. State of infection report – Q2 2014. http://landing.damballa.com/state-infections-report-q2-2014.html, 2014.

[36] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. *IACR Cryptology ePrint Archive*, 2008:538, 2008.

[37] Brian Dawkins. Siobhan's problem: the coupon collector revisited. *The American Statistician*, 45(1):76–82, 1991.

[38] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data*, pages 185–198. ACM, 2016.

[39] John R Douceur, Atul Adya, William J Bolosky, P Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 617–624. IEEE, 2002.

[40] Yitao Duan. Distributed key generation for encrypted deduplication: Achieving the strongest privacy. In *Proceedings of the ACM Workshop on Cloud Computing Security*, pages 57–68. ACM, 2014.

[41] Mike Dutch. Understanding data deduplication ratios. In *SNIA Data Management Forum*, page 7, 2008.

[42] Eupedia. http://www.eupedia.com/genetics/medical_dna_test.shtml, accessed in October 2015.

[43] FDA. Biomarker qualification program. http://www.fda.gov/Drugs/DevelopmentApprovalProcess/DrugDevelopmentToolsQualificationProgram/ucm284076.htm, accessed in October 2015.

[44] FSL. Traces and snapshots public archive. http://tracer.filesystems.org, 2012.

[45] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. Hardidx: Practical and secure index with sgx. In *Proceedings of IFIP Annual Conference on Data and Applications Security and Privacy*, pages 386–408. Springer, 2017.

[46] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[47] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In *Advances in Cryptology–EUROCRYPT 2014*, pages 405–422. Springer, 2014.

[48] Eu-Jin Goh et al. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[49] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the 19th annual ACM symposium on Theory of computing*, pages 182–194. ACM, 1987.

[50] Philippe Golle, Jessica Staddon, and Brent Waters. Secure conjunctive keyword search over encrypted data. In *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security*, volume 4, pages 31–45. Springer, 2004.

[51] David Grawrock. *Dynamics of a Trusted Platform: A building block approach*. Intel Press, 2009.

[52] Trusted Computing Group. TPM main specification. http://www.trustedcomputinggroup.org/tpm-main-specification/, 2016.

[53] Shay Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptology ePrint Archive*, 2016:204, 2016.

[54] Debayan Gupta, Benjamin Mood, Joan Feigenbaum, Kevin Butler, and Patrick Traynor. Using Intel software guard extensions for efficient two-party secure function evaluation. In *Proceedings of the 2016 International Conference on Financial Cryptography and Data Security*, pages 302–318. Springer, 2016.

[55] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 491–500. ACM, 2011.

[56] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy*, 8(6):40–47, 2010.

[57] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, page 11. ACM, 2013.

[58] Zhicong Huang, Erman Ayday, Jacques Fellay, Jean-Pierre Hubaux, and Ari Juels. GenoGuard: Protecting genomic data against brute-force attacks. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2015.

[59] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proceedings of Network and Distributed System Security*, volume 20, page 12. Internet Society, 2012.

[60] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel SGX: EPID provisioning and attestation services. https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services, 2016.

[61] Michal Kaczmarczyk, Marcin Barczynski, Wojciech Kilian, and Cezary Dubnicki. Reducing impact of data fragmentation caused by in-line deduplication. In *Proceedings of the 5th Annual International Systems and Storage Conference*, page 15. ACM, 2012.

[62] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Proceedings of Financial Cryptography*, pages 258–274. Springer, 2013.

[63] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM SIGSAC Conference on Computer and Communications Security*, pages 965–976. ACM, 2012.

[64] David Kaplan, Jeremy Powell, and Tom Woller. AMD memory encryption. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, 2016.

[65] KeywordDiscovery. Keyword and search engines statistics. https://www.keyworddiscovery.com/keyword-stats.html?date=2017-04-01, 2017.

[66] Kaoru Kurosawa and Yasuhiro Ohtaki. Uc-secure searchable symmetric encryption. In *Proceedings of Financial Cryptography*, volume 7397, pages 285–298. Springer, 2012.

[67] Kaoru Kurosawa and Yasuhiro Ohtaki. How to update documents verifiably in searchable symmetric encryption. In *Proceedings of International Conference on Cryptology and Network Security*, pages 309–328. Springer, 2013.

[68] Jin Li, Xiaofeng Chen, Mingqiang Li, Jingwei Li, Patrick PC Lee, and Wenjing Lou. Secure deduplication with efficient and reliable convergent key management. *IEEE transactions on Parallel and Distributed Systems*, 25(6):1615–1625, 2014.

[69] Jingwei Li, Chuan Qin, Patrick PC Lee, and Xiaosong Zhang. Information leakage in encrypted deduplication via frequency analysis. In *Proceedings of The 47th IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 2110–2118. IEEE, 2017.

[70] Mark Lillibridge, Kave Eshghi, and Deepavali Bhagwat. Improving restore speed for backup systems that use inline chunk-based deduplication. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, pages 183–198. USENIX Association, 2013.

[71] Chuanyi Liu, Yingping Lu, Chunhui Shi, Guanlin Lu, David HC Du, and Dong-Sheng Wang. ADMAD: Application-driven metadata aware de-duplication archival storage system. In *Proceedings of the 5th IEEE International Workshop on Storage Network Architecture and Parallel I/Os*, pages 29–35. IEEE, 2008.

[72] Jian Liu, N Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 874–885. ACM, 2015.

[73] Guanlin Lu, Yu Jin, and David HC Du. Frequency based chunking for data de-duplication. In *Proceedings of 2010 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 287–296. IEEE, 2010.

[74] Yanbin Lu. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *Proceedings of Network and Distributed System Security*. Internet Society, 2012.

[75] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 190–200. ACM, 2005.

[76] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[77] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, page 10. ACM, 2013.

[78] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (TOS)*, 7(4):14, 2012.

[79] Young Jin Nam, Dongchul Park, and David HC Du. Assuring demanded read performance of data deduplication storage with backup datasets. In *Proceedings of IEEE the 20th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 201–208. IEEE, 2012.

[80] National Human Genome Research Institute. http://www.genome.gov/sequencingcosts/, accessed in October 2015.

[81] Muhammad Naveed. The fallacy of composition of oblivious ram and searchable encryption. *IACR Cryptology ePrint Archive*, 2015:668, 2015.

[82] Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang. Privacy in the genomic era. *ACM Computing Surveys*, 48(1):6, 2015.

[83] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[84] Matti Niemenmaa, Aleksi Kallio, André Schumacher, Petri Klemelä, Eija Korpelainen, and Keijo Heljanko. Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28(6):876–877, 2012.

[85] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *Proceedings of the 2016 USENIX Security Symposium*, pages 619–636. USENIX Association, 2016.

[86] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *Advances in Cryptology–CRYPTO 2011*, pages 91–110. Springer, 2011.

[87] João Paulo and José Pereira. A survey and classification of storage deduplication systems. *ACM Computing Surveys*, 47(1):11, 2014.

[88] Nelly Porter, Jason Garms, and Sergey Simakov. Introducing Asylo: An open-source framework for confidential computing. `https://cloudplatform.googleblog.com/2018/05/Introducing-Asylo-an-open-source-framework-for-confidential-computing.html`, 2018.

[89] Franco P Preparata and Dilip V Sarwate. Computational complexity of fourier transforms over finite fields. *Mathematics of Computation*, 31(139):740–751, 1977.

[90] Pasquale Puzio, Refik Molva, Melek Onen, and Sergio Loureiro. ClouDedup: Secure deduplication with encrypted data for cloud storage. In *Proceedings of IEEE the 5th International Conference on Cloud Computing Technology and Science*, volume 1, pages 363–370. IEEE, 2013.

[91] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: Closing digital side-channels through obfuscated execution. In *Proceedings of the 2015 USENIX Security Symposium*, pages 431–446. USENIX Association, 2015.

[92] Dan M Roden, Jill M Pulley, Melissa A Basford, Gordon R Bernard, Ellen W Clayton, Jeffrey R Balser, and Dan R Masys. Development of a large-scale de-identified DNA biobank to enable personalized medicine. *Clinical Pharmacology & Therapeutics*, 84 (3):362–369, 2008.

[93] SAMTools. Sequcence alignment/map format specification. https://samtools.github.io/hts-specs/SAMv1.pdf, accessed in September 2015.

[94] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Using ARM TrustZone to build a trusted language runtime for mobile applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 67–80. ACM, 2014.

[95] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 38–54. IEEE, 2015.

[96] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Proceedings of Theory of Cryptography Conference*, pages 457–473. Springer, 2009.

[97] Elaine Shi, John Bethencourt, TH Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 350–364. IEEE, 2007.

[98] Suyash S Shringarpure and Carlos D Bustamante. Privacy risks from genomic data-sharing beacons. *The American Journal of Human Genetics*, 97(5):631–646, 2015.

[99] SkyhighNetworks. Skyhigh for salesforce. https://www.skyhighnetworks.com/product/salesforce-encryption/, 2017.

[100] Dawn Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 44–55. IEEE, 2000.

[101] Kiran Srinivasan, Timothy Bisson, Garth R Goodson, and Kaladhar Voruganti. iDedup: Latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, pages 1–4. USENIX Association, 2012.

[102] Standord University. Pairing-based cryptography libray. http://crypto.stanford.edu/pbc/, accessed in May 2014.

[103] Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In *Proceedings of International Conference on Financial Cryptography and Data Security*, pages 99–118. Springer, 2014.

[104] Richard P Stanley. What is enumerative combinatorics? In *Enumerative Combinatorics*, pages 1–63. Springer, 1986.

[105] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *Proceedings of Network and Distributed System Security*, volume 14, pages 23–26. Internet Society, 2014.

[106] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y. Thomas Hou, and Hui Li. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In *Proceedings of ACM Asia Conference on Computer and Communications Security*, pages 71–82. ACM, 2013.

[107] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y Thomas Hou, and Hui Li. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel and Distributed Systems*, 25 (11):3025–3035, 2014.

[108] Wenhai Sun, Shucheng Yu, Wenjing Lou, Y Thomas Hou, and Hui Li. Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In *Proceedings of IEEE Conference on Computer Communications*, pages 226–234. IEEE, 2014.

[109] Wenhai Sun, Xuefeng Liu, Wenjing Lou, Y Thomas Hou, and Hui Li. Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In *Proceedings of IEEE Conference on Computer Communications*, pages 2110–2118. IEEE, 2015.

[110] Wenhai Sun, Shucheng Yu, Wenjing Lou, Y Thomas Hou, and Hui Li. Protecting your right: verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1187–1198, 2016.

[111] Wenhai Sun, Ning Zhang, Wenjing Lou, and Y Thomas Hou. When gene meets cloud: Enabling scalable and efficient range query on encrypted genomic data. In *Proceedings of IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[112] Wenhai Sun, Ning Zhang, Wenjing Lou, and Y Thomas Hou. Tapping the potential: Secure chunk-based deduplication of encrypted data for cloud backup. In *Proceedings of IEEE Conference on Communications and Network Security*. IEEE, 2018.

[113] Wenhai Sun, Ruide Zhang, Wenjing Lou, and Y Thomas Hou. Rearguard: Secure keyword search using trusted hardware. In *Proceedings of IEEE Conference on Computer Communications*, pages 801–809. IEEE, 2018.

[114] Jaikumar Vijayan. Cloud security concerns are overblown, experts say. http://www.computerworld.com/s/article/9246632/Cloud_security_concerns_are_overblown_experts_say, 2014.

[115] Boyang Wang, Yantian Hou, Ming Li, Haitao Wang, and Hui Li. Maple: scalable multi-dimensional range search over encrypted cloud data with tree-based index. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 111–122. ACM, 2014.

[116] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions on parallel and distributed systems*, 23(8):1467–1479, 2012.

[117] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 492–503. ACM, 2015.

[118] Tom White. The small files problem. http://blog.cloudera.com/blog/2009/02/the-small-files-problem/, accessed in November 2015.

[119] Jia Xu, Ee-Chien Chang, and Jianying Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pages 195–206. ACM, 2013.

[120] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 640–656. IEEE, 2015.

[121] Lauren J. Young. Genomic data growing faster than twitter and youtube. https://spectrum.ieee.org/tech-talk/biomedical/diagnostics/the-human-os-is-at-the-top-of-big-data, accessed in October 2015.

[122] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, pages 270–282. ACM, 2016.

[123] Ye Zhang, Chun Jason Xue, Duncan S Wong, Nikos Mamoulis, and Siu Ming Yiu. Acceleration of composite order bilinear pairing on graphics hardware. In *Proceedings of International Conference on Information and Communications Security*, pages 341–348. Springer, 2012.

[124] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *Proceedings of the 2016 USENIX Security Symposium*, pages 707–720. USENIX Association, 2016.